Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems

Sergey Levine[1,2], **Aviral Kumar**[1], **George Tucker**[2], **Justin Fu**[1]
[1]UC Berkeley, [2]Google Research, Brain Team

Abstract

In this tutorial article, we aim to provide the reader with the conceptual tools needed to get started on research on offline reinforcement learning algorithms: reinforcement learning algorithms that utilize previously collected data, without additional online data collection. Offline reinforcement learning algorithms hold tremendous promise for making it possible to turn large datasets into powerful decision making engines. Effective offline reinforcement learning methods would be able to extract policies with the maximum possible utility out of the available data, thereby allowing automation of a wide range of decision-making domains, from healthcare and education to robotics. However, the limitations of current algorithms make this difficult. We will aim to provide the reader with an understanding of these challenges, particularly in the context of modern deep reinforcement learning methods, and describe some potential solutions that have been explored in recent work to mitigate these challenges, along with recent applications, and a discussion of perspectives on open problems in the field.

1 Introduction

Reinforcement learning provides a mathematical formalism for learning-based control. By utilizing reinforcement learning, we can automatically acquire near-optimal behavioral skills, represented by policies, for optimizing user-specified reward functions. The reward function defines *what* an agent should do, and a reinforcement learning algorithm determines *how* to do it. While the reinforcement learning algorithms have been an active area of research for decades, the introduction of effective high-capacity function approximators – deep neural networks – into reinforcement learning, along with effective algorithms for training them, has allowed reinforcement learning methods to attain excellent results along a wide range of domains (Tesauro, 1994; Hafner and Riedmiller, 2011; Levine and Koltun, 2013; Mnih et al., 2013; Levine et al., 2016; Silver et al., 2017; Kalashnikov et al., 2018).

However, the fact that reinforcement learning algorithms provide a fundamentally *online* learning paradigm is also one of the biggest obstacles to their widespread adoption. The process of reinforcement learning involves iteratively collecting experience by interacting with the environment, typically with the latest learned policy, and then using that experience to improve the policy (Sutton and Barto, 1998). In many settings, this sort of online interaction is impractical, either because data collection is expensive (e.g., in robotics, educational agents, or healthcare) and dangerous (e.g., in autonomous driving, or healthcare). Furthermore, even in domains where online interaction is feasible, we might still prefer to utilize previously collected data instead – for example, if the domain is complex and effective generalization requires large datasets.

Indeed, the success of machine learning methods across a range of practically relevant problems over the past decade can in large part be attributed to the advent of scalable *data-driven* learning methods, which become better and better as they are trained with more data. Online reinforcement learning is difficult to reconcile with this paradigm. While this was arguably less of an issue when reinforcement learning methods utilized low-dimensional or linear parameterizations, and therefore relied on small datasets for small problems that were easy to collect or simulate (Lange et al., 2012), once deep networks are incorporated into reinforcement learning, it is tempting to consider whether the same kind of data-driven learning can be applied with reinforcement learning objectives, thus resulting in *data-driven reinforcement learning* that utilizes only previously collected offline data, without any

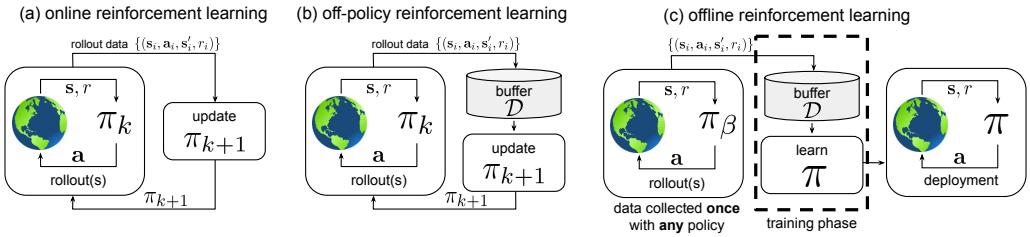

Figure 1: Pictorial illustration of classic online reinforcement learning (a), classic off-policy reinforcement learning (b), and offline reinforcement learning (c). In online reinforcement learning (a), the policy π_k is updated with streaming data collected by π_k itself. In the classic off-policy setting (b), the agent's experience is appended to a data buffer (also called a replay buffer) $\mathcal{D}$, and each new policy π_k collects additional data, such that $\mathcal{D}$ is composed of samples from $\pi_0, \pi_1, \ldots, \pi_k$, and all of this data is used to train an updated new policy π_{k+1}. In contrast, offline reinforcement learning employs a dataset $\mathcal{D}$ collected by some (potentially unknown) behavior policy π_β. The dataset is collected once, and is not altered during training, which makes it feasible to use large previous collected datasets. The training process does not interact with the MDP at all, and the policy is only deployed after being fully trained.

additional online interaction (Kumar, 2019; Fu et al., 2020). See Figure 1 for a pictorial illustration. A number of recent works have illustrated the power of such an approach in enabling data-driven learning of policies for dialogue (Jaques et al., 2019), robotic manipulation behaviors (Ebert et al., 2018; Kalashnikov et al., 2018), and robotic navigation skills (Kahn et al., 2020).

Unfortunately, such data-driven *offline* reinforcement learning also poses major algorithmic challenges. As we will discuss in this article, many commonly used reinforcement learning methods can learn from off-policy data, but such methods often cannot learn effectively from entire offline data, without any additional on-policy interaction. High-dimensional and expressive function approximation generally exacerbates this issue, since function approximation leaves the algorithms vulnerable to distributional shift, one of the central challenges with offline reinforcement learning. However, the appeal of a fully offline reinforcement learning framework is enormous: in the same way that supervised machine learning methods have enabled data to be turned into generalizable and powerful *pattern recognizers* (e.g., image classifiers, speech recognition engines, etc.), offline reinforcement learning methods equipped with powerful function approximation may enable data to be turned into generalizable and powerful *decision making engines*, effectively allowing anyone with a large enough dataset to turn this dataset into a policy that can optimize a desired utility criterion. From healthcare decision-making support to autonomous driving to robotics, the implications of a reliable and effective offline reinforcement learning method would be immense.

In some application domains, the lack of effective offline reinforcement learning methods has driven research in a number of interesting directions. For example, in robotics and autonomous driving, a rapidly growing research topic is the study of simulation to real-world transfer: training policies with reinforcement learning in simulation and then transferring these policies into the real world (Sadeghi and Levine, 2017; Tan et al., 2018; Chebotar et al., 2019). While this approach is very pragmatic (and often effective), its popularity highlights the deficiency in offline reinforcement learning methods: if it was possible to simply train policies with previously collected data, it would likely be unnecessary in many cases to manually design high-fidelity simulators for simulation-to-real-world transfer. After all, outside of reinforcement learning (e.g., in computer vision, NLP, or speech recognition), transfer from simulation is comparatively much less prevalent, since data-driven learning is so effective.

The goal of this article is to provide the reader with the conceptual tools needed to get started on research in the field of offline reinforcement learning (also called batch reinforcement learning (Ernst et al., 2005; Riedmiller, 2005; Lange et al., 2012)), so as to hopefully begin addressing some of these deficiencies. To this end, we will present the offline reinforcement learning problem formulation, and describe some of the challenges associated with this problem setting, particularly in light of recent research on deep reinforcement learning and the interaction between reinforcement learning and high-dimensional function approximator, such as deep networks. We will cover a variety of offline reinforcement learning methods studied in the literature. For each one, we will discuss the conceptual challenges, and initial steps taken to mitigate these challenges. We will then discuss some of the applications of offline reinforcement learning techniques that have already been explored, despite

the limitations of current methods, and conclude with some perspectives on future work and open problems in the field.

2 Offline Reinforcement Learning Problem Statement and Overview

In this section, we will introduce the mathematical formalism of reinforcement learning and define our notation, and then set up the offline reinforcement learning problem setting, where the goal is to learn near-optimal policies from previously collected data. Then, we will briefly discuss some of the intuition behind why the offline reinforcement learning problem setting poses some unique challenges, using a supervised behavioral cloning example.

2.1 Reinforcement Learning Preliminaries

In this section, we will define basic reinforcement learning concepts, following standard textbook definitions (Sutton and Barto, 1998). Reinforcement learning addresses the problem of learning to control a dynamical system, in a general sense. The dynamical system is fully defined by a fully-observed or partially-observed Markov decision process (MDP).

Definition 2.1 (Markov decision process)**.** The Markov decision process is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$, where $\mathcal{S}$ is a set of states $\mathbf{s} \in \mathcal{S}$, which may be either discrete or continuous (i.e., multi-dimensional vectors), $\mathcal{A}$ is a set of actions $\mathbf{a} \in \mathcal{A}$, which similarly can be discrete or continuous, T defines a conditional probability distribution of the form $T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ that describes the dynamics of the system,[1] d_0 defines the initial state distribution $d_0(\mathbf{s}_0)$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines a reward function, and $\gamma \in (0, 1]$ is a scalar discount factor.

We will use the fully-observed formalism in most of this article, though the definition for the partially observed Markov decision process (POMDP) is also provided for completeness. The MDP definition can be extended to the partially observed setting as follows:

Definition 2.2 (Partially observed Markov decision process)**.** The partially observed Markov decision process is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T, d_0, E, r, \gamma)$, where $\mathcal{S}$, $\mathcal{A}$, T, d_0, r, and γ are defined as before, $\mathcal{O}$ is a set of observations, where each observation is given by $\mathbf{o} \in \mathcal{O}$, and E is an emission function, which defines the distribution $E(\mathbf{o}_t|\mathbf{s}_t)$.

The final goal in a reinforcement learning problem is to learn a policy, which defines a distribution over actions conditioned on states, $\pi(\mathbf{a}_t|\mathbf{s}_t)$, or conditioned on observations in the partially observed setting, $\pi(\mathbf{a}_t|\mathbf{o}_t)$. The policy may also be conditioned on an observation history, $\pi(\mathbf{a}_t|\mathbf{o}_{0:t})$. From these definitions, we can derive the *trajectory distribution*. The trajectory is a sequence of states and actions of length H, given by $\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$, where H may be infinite. The trajectory distribution p_π for a given MDP $\mathcal{M}$ and policy π is given by

$$p_\pi(\tau) = d_0(\mathbf{s}_0) \prod_{t=0}^{H} \pi(\mathbf{a}_t|\mathbf{s}_t) T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t).$$

This definition can easily be extended into the partially observed setting by including the observations $\mathbf{o}_t$ and emission function $E(\mathbf{o}_t|\mathbf{s}_t)$. The reinforcement learning objective, $J(\pi)$, can then be written as an expectation under this trajectory distribution:

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right]. \tag{1}$$

When H is infinite, it is sometimes also convenient to assume that the Markov chain on $(\mathbf{s}_t, \mathbf{a}_t)$ defined by $\pi(\mathbf{a}_t|\mathbf{s}_t)T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ is ergodic, and define the objective in terms of the expected reward under the stationary distribution of this Markov chain (Sutton and Barto, 1998). This definition is somewhat complicated by the role of the discount factor. For a full discussion of this topic, we refer the reader to prior work (Thomas, 2014).

[1] We will sometimes use time subscripts (i.e., $\mathbf{s}_{t+1}$ follows $\mathbf{s}_t$), and sometimes "prime" notation (i.e., $\mathbf{s}'$ is the state that follows $\mathbf{s}$). Explicit time subscripts can help clarify the notation in finite-horizon settings, while "prime" notation is simpler in infinite-horizon settings where absolute time step indices are less meaningful.

In many cases, we will find it convenient to refer to the marginals of the trajectory distribution $p_\pi(\tau)$. We will use $d^\pi(\mathbf{s})$ to refer to the overall state visitation frequency, averaged over the time steps, and $d_t^\pi(\mathbf{s}_t)$ to refer to the state visitation frequency at time step t.

In this section, we will briefly summarize different types of reinforcement learning algorithms and present definitions. At a high level, all standard reinforcement learning algorithms follow the same basic learning loop: the agent *interacts* with the MDP $\mathcal{M}$ by using some sort of *behavior policy*, which may or may not match $\pi(\mathbf{a}|\mathbf{s})$, by observing the current state $\mathbf{s}_t$, selecting an action $\mathbf{a}_t$, and then observing the resulting next state $\mathbf{s}_{t+1}$ and reward value $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$. This may repeat for multiple steps, and the agent then uses the observed transitions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_t)$ to update its policy. This update might also utilize previously observed transitions. We will use $\mathcal{D} = \{(\mathbf{s}_t^i, \mathbf{a}_t^i, \mathbf{s}_{t+1}^i, r_t^i)\}$ to denote the set of transitions that are available for the agent to use for updating the policy ("learning"), which may consist of either all transitions seen so far, or some subset thereof.

Policy gradients. One of the most direct ways to optimize the RL objective in Equation 1 is to directly estimate its gradient. In this case, we typically assume that the policy is parameterized by a parameter vector θ, and therefore given by $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$. For example, θ might denote the weights of a deep network that outputs the logits for the (discrete) actions $\mathbf{a}_t$. In this case, we can express the gradient of the objective with respect to θ as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_{t=0}^{H} \gamma^t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \underbrace{\left(\sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - b(\mathbf{s}_t) \right)}_{\text{return estimate } \hat{A}(\mathbf{s}_t, \mathbf{a}_t)} \right], \tag{2}$$

where the return estimator $\hat{A}(\mathbf{s}_t, \mathbf{a}_t)$ can itself be learned as a separate neural network *critic*, as discussed below, or it can simply be estimated with Monte Carlo samples, in which case we simply generate samples from $p_{\pi_\theta}(\tau)$, and then sum up the rewards over the time steps of the sampled trajectory. The baseline $b(\mathbf{s}_t)$ can be estimated as the average reward over the sampled trajectories, or by using a value function estimator $V(\mathbf{s}_t)$, which we discuss in the dynamic programming section. We can equivalently write this gradient expression as an expectation with respect to $d_t^\pi(\mathbf{s}_t)$ as

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{H} \mathbb{E}_{\mathbf{s}_t \sim d_t^\pi(\mathbf{s}_t), \mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\gamma^t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \hat{A}(\mathbf{s}_t, \mathbf{a}_t) \right].$$

A common modification is to drop the γ^t term in front of the gradient, which approximates an average reward setting (Thomas, 2014). Dropping this term and adopting an infinite-horizon formulation, we can further rewrite the policy gradient as expectation under $d^\pi(\mathbf{s})$ as

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1-\gamma} \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s}_t), \mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})} \left[\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}(\mathbf{s}, \mathbf{a}) \right].$$

The constant scaling term $\frac{1}{1-\gamma}$ is often disregarded. This infinite-horizon formulation is often convenient to work with for analyzing and deriving policy gradient methods. For a full derivation of this gradient, we refer the reader to prior work (Sutton et al., 2000; Kakade, 2002; Schulman et al., 2015). We can summarize a basic Monte Carlo policy gradient algorithm as follows:

Algorithm 1 On-policy policy gradient with Monte Carlo estimator

1: initialize θ_0
2: **for** iteration $k \in [0, \ldots, K]$ **do**
3: sample trajectories $\{\tau_i\}$ by running $\pi_{\theta_k}(\mathbf{a}_t|\mathbf{s}_t)$ ▷ each τ_i consists of $\mathbf{s}_{i,0}, \mathbf{a}_{i,0}, \ldots, \mathbf{s}_{i,H}, \mathbf{a}_{i,H}$
4: compute $\mathcal{R}_{i,t} = \sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$
5: fit $b(\mathbf{s}_t)$ to $\{\mathcal{R}i, t\}$ ▷ use constant $b_t = \frac{1}{N} \sum_i \mathcal{R}i, t$, or fit $b(\mathbf{s}_t)$ to $\{\mathcal{R}i, t\}$
6: compute $\hat{A}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) = \mathcal{R}_{i,t} - b(\mathbf{s}_t)$
7: estimate $\nabla_{\theta_k} J(\pi_{\theta_k}) \approx \sum_{i,t} \nabla_{\theta_k} \log \pi_{\theta_k}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{A}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$
8: update parameters: $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta_k} J(\pi_{\theta_k})$
9: **end for**

For additional details on standard on-policy policy gradient methods, we refer the reader to prior work (Sutton et al., 2000; Kakade, 2002; Schulman et al., 2015).

Approximate dynamic programming. Another way to optimize the reinforcement learning objective is to observe that, if we can accurately estimate a state or state-action *value function*, it is easy to then recover a near-optimal policy. A value function provides an estimate of the expected cumulative reward that will be obtained by following some policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ when starting from a given state $\mathbf{s}_t$, in the case of the state-value function $V^\pi(\mathbf{s}_t)$, or when starting from a state-action tuple $(\mathbf{s}_t, \mathbf{a}_t)$, in the case of the state-action value function $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$. We can define these value functions as:

$$V^\pi(\mathbf{s}_t) = \mathbb{E}_{\tau \sim p_\pi(\tau|\mathbf{s}_t)} \left[\sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\tau \sim p_\pi(\tau|\mathbf{s}_t, \mathbf{a}_t)} \left[\sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_t, \mathbf{a}_t) \right].$$

From this, we can derive recursive definitions for these value functions, which are given as

$$V^\pi(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)} \left[Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \right]$$
$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[V^\pi(\mathbf{s}_{t+1}) \right].$$

We can combine these two equations to express the $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ in terms of $Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$:

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_{t+1} \sim \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})} \left[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})) \right]. \tag{3}$$

We can also express these in terms of the *Bellman operator* for the policy π, which we denote $\mathcal{B}^\pi$. For example, Equation (3) can be written as $\vec{Q}^\pi = \mathcal{B}^\pi \vec{Q}^\pi$, where $\vec{Q}^\pi$ denotes the Q-function Q^π represented as a vector of length $|\mathcal{S}| \times |\mathcal{A}|$. Before moving on to deriving learning algorithms based on these definitions, we briefly discuss some properties of the Bellman operator. This Bellman operator has a unique fixed point that corresponds to the true Q-function for the policy $\pi(\mathbf{a}|\mathbf{s})$, which can be obtained by repeating the iteration $\vec{Q}^\pi_{k+1} = \mathcal{B}^\pi \vec{Q}^\pi_k$, and it can be shown that $\lim_{k \to \infty} \vec{Q}^\pi_k = \vec{Q}^\pi$, which obeys Equation (3) (Sutton and Barto, 1998). The proof for this follows from the observation that $\mathcal{B}^\pi$ is a contraction in the ℓ_∞ norm (Lagoudakis and Parr, 2003).

Based on these definitions, we can derive two commonly used algorithms based on dynamic programming: Q-learning and actor-critic methods. To derive Q-learning, we express the policy implicitly in terms of the Q-function, as $\pi(\mathbf{a}_t|\mathbf{s}_t) = \delta(\mathbf{a}_t = \arg\max Q(\mathbf{s}_t, \mathbf{a}_t))$, and only learn the Q-function $Q(\mathbf{s}_t, \mathbf{a}_t)$. By substituting this (implicit) policy into the above dynamic programming equation, we obtain the following condition on the optimal Q-function:

$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]. \tag{4}$$

We can again express this as $\vec{Q} = \mathcal{B}^\star \vec{Q}$ in vector notation, where $\mathcal{B}^\star$ now refers to the Bellman optimality operator. Note however that this operator is not linear, due to the maximization on the right-hand side in Equation (4). To turn this equation into a learning algorithm, we can minimize the difference between the left-hand side and right-hand side of this equation with respect to the parameters of a parametric Q-function estimator with parameters ϕ, $Q_\phi(\mathbf{s}_t, \mathbf{a}_t)$. There are a number of variants of this Q-learning procedure, including variants that fully minimize the difference between the left-hand side and right-hand side of the above equation at each iteration, commonly referred to as fitted Q-iteration (Ernst et al., 2005; Riedmiller, 2005), and variants that take a single gradient step, such as the original Q-learning method (Watkins and Dayan, 1992). The commonly used variant in deep reinforcement learning is a kind of hybrid of these two methods, employing a replay buffer (Lin, 1992) and taking gradient steps on the Bellman error objective concurrently with data collection (Mnih et al., 2013). We write out a general recipe for Q-learning methods in Algorithm 2.

Classic Q-learning can be derived as the limiting case where the buffer size is 1, and we take $G = 1$ gradient steps and collect $S = 1$ transition samples per iteration, while classic fitted Q-iteration runs the inner gradient descent phase to convergence (i.e., $G = \infty$), and uses a buffer size equal to the number of sampling steps S. Note that many modern implementations also employ a *target network*, where the target value $r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')$ actually uses ϕ_L, where L is a lagged iteration (e.g., the last k that is a multiple of 1000). Note that these approximations violate the assumptions under which Q-learning algorithms can be proven to converge. However, recent work suggests that high-capacity function approximators, which correspond to a very large set $\mathcal{Q}$, generally do tend to make this method convergent in practice, yielding a Q-function that is close to $\vec{Q}^\pi$ (Fu et al., 2019; Van Hasselt et al., 2018).

Algorithm 2 Generic Q-learning (includes FQI and DQN as special cases)

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
3: initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
4: initialize $\mathbf{s} \sim d_0(\mathbf{s})$
5: **for** iteration $k \in [0, \ldots, K]$ **do**
6: **for** step $s \in [0, \ldots, S-1]$ **do**
7: $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy
8: $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ▷ sample next state from MDP
9: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big
10: **end for**
11: $\phi_{k,0} \leftarrow \phi_k$
12: **for** gradient step $g \in [0, \ldots, G-1]$ **do**
13: sample batch $B \subset \mathcal{D}$ ▷ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$
14: estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$
15: update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha\nabla_{\phi_{k,g}}\mathcal{E}(B, \phi_{k,g})$
16: **end for**
17: $\phi_{k+1} \leftarrow \phi_{k,G}$ ▷ update parameters
18: **end for**

Actor-critic algorithms. Actor-critic algorithms combine the basic ideas from policy gradients and approximate dynamic programming. Such algorithms employ *both* a parameterized policy and a parameterized value function, and use the value function to provide a better estimate of $\hat{A}(\mathbf{s}, \mathbf{a})$ for policy gradient calculation. There are a number of different variants of actor-critic methods, including on-policy variants that directly estimate $V^\pi(\mathbf{s})$ (Konda and Tsitsiklis, 2000), and off-policy variants that estimate $Q^\pi(\mathbf{s}, \mathbf{a})$ via a parameterized state-action value function $Q^\pi_\phi(\mathbf{s}, \mathbf{a})$ (Haarnoja et al., 2018, 2017; Heess et al., 2015). We will focus on the latter class of algorithms, since they can be extended to the offline setting. The basic design of such an algorithm is a straightforward combination of the ideas in dynamic programming and policy gradients. Unlike Q-learning, which directly attempts to learn the optimal Q-function, actor-critic methods aim to learn the Q-function corresponding to the current parameterized policy $\pi_\theta(\mathbf{a}|\mathbf{s})$, which must obey the equation

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim T(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),\mathbf{a}_{t+1}\sim\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})}\left[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right].$$

As before, this equation can be expressed in vector form in terms of the Bellman operator for the policy, $\vec{Q}^\pi = \mathcal{B}^\pi\vec{Q}^\pi$, where $\vec{Q}^\pi$ denotes the Q-function Q^π represented as a vector of length $|\mathcal{S}| \times |\mathcal{A}|$. We can now instantiate a complete algorithm based on this idea, shown in Algorithm 3.

For more details, we refer the reader to standard textbooks and prior works (Sutton and Barto, 1998; Konda and Tsitsiklis, 2000). Actor-critic algorithms are closely related with another class of methods that frequently arises in dynamic programming, called policy iteration (PI) (Lagoudakis and Parr, 2003). Policy iteration consists of two phases: policy evaluation and policy improvement. The policy evaluation phase computes the Q-function for the current policy π, Q^π, by solving for the fixed point such that $Q^\pi = \mathcal{B}^\pi Q^\pi$. This can be done via linear programming or solving a system of linear equations, as we will discuss in Section 4, or via gradient updates, analogously to line 15 in Algorithm 3. The next policy iterate is then computed in the policy improvement phase, by choosing the action that greedily maximizes the Q-value at each state, such that $\pi_{k+1}(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q^{\pi_k}(\mathbf{s}, \mathbf{a}))$, or by using a gradient based update procedure as is employed in Algorithm 3 on line 24. In the absence of function approximation (e.g., with tabular representations) policy iteration produces a monotonically improving sequence of policies, and converges to the optimal policy. Policy iteration can be obtained as a special case of the generic actor-critic algorithm in Algorithm 3 when we set $G_Q = \infty$ and $G_\pi = \infty$, when the buffer $\mathcal{D}$ consists of each and every transition of the MDP.

Model-based reinforcement learning. Model-based reinforcement learning is a general term that refers to a broad class of methods that utilize explicit estimates of the transition or dynamics function $T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, parameterized by a parameter vector ψ, which we will denote $T_\psi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. There is no single recipe for a model-based reinforcement learning method. Some commonly used model-based reinforcement learning algorithms learn only the dynamics model $T_\psi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, and then

Algorithm 3 Generic off-policy actor-critic

initialize ϕ_0
initialize θ_0
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0, \ldots, K]$ **do**
  for step $s \in [0, \ldots, S-1]$ **do**
    $\mathbf{a} \sim \pi_{\theta_k}(\mathbf{a}|\mathbf{s})$ ▷ sample action from current policy
    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ▷ sample next state from MDP
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big
  end for
  $\phi_{k,0} \leftarrow \phi_k$
  for gradient step $g \in [0, \ldots, G_Q - 1]$ **do**
    sample batch $B \subset \mathcal{D}$ ▷ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$
    estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi_k(\mathbf{a}'|\mathbf{s}')} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$
    update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha_Q \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$
  end for
  $\phi_{k+1} \leftarrow \phi_{k,G_Q}$ ▷ update Q-function parameters
  $\theta_{k,0} \leftarrow \theta_k$
  for gradient step $g \in [0, \ldots G_\pi - 1]$ **do**
    sample batch of states $\{\mathbf{s}_i\}$ from $\mathcal{D}$
    for each $\mathbf{s}_i$, sample $\mathbf{a}_i \sim \pi_{\theta_{k,g}}(\mathbf{a}|\mathbf{s}_i)$ ▷ do not use actions in the buffer!
    for each $(\mathbf{s}_i, \mathbf{a}_i)$, compute $\hat{A}(\mathbf{s}_i, \mathbf{a}_i) = Q_{\phi_{k+1}}(\mathbf{s}_i, \mathbf{a}_i) - \mathbb{E}_{\mathbf{a} \sim \pi_{k,g}(\mathbf{a}|\mathbf{s}_i)}[Q_{\phi_{k+1}}(\mathbf{s}_i, \mathbf{a})]$
    $\nabla_{\theta_{k,g}} J(\pi_{\theta_{k,g}}) \approx \frac{1}{N} \nabla_{\theta_{k,g}} \log \pi_{\theta_{k,g}}(\mathbf{s}_i, \mathbf{a}_i) \hat{A}(\mathbf{s}_i, \mathbf{a}_i)$
    $\theta_{k,g+1} \leftarrow \theta_{k,g} + \alpha_\pi \nabla_{\theta_{k,g}} J(\pi_{\theta_{k,g}})$
  end for
  $\theta k + 1 \leftarrow \theta_{k,G_\pi}$ ▷ update policy parameters
end for

utilize it for planning at test time, often by means of model-predictive control (MPC) (Tassa et al., 2012) with various trajectory optimization methods (Nagabandi et al., 2018; Chua et al., 2018). Other model-based reinforcement learning methods utilize a learned policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ in addition to the dynamics model, and employ backpropagation through time to optimize the policy with respect to the expected reward objective (Deisenroth and Rasmussen, 2011). Yet another set of algorithms employ the model to generate "synthetic" samples to augment the sample set available to model-free reinforcement learning methods. The classic Dyna algorithm uses this recipe in combination with Q-learning and one-step predictions via the model from previously seen states (Sutton, 1991), while a variety of recently proposed algorithms employ synthetic model-based rollouts with policy gradients (Parmas et al., 2019; Kaiser et al., 2019a) and actor-critic algorithms (Janner et al., 2019). Since there are so many variants of model-based reinforcement learning algorithms, we will not go into detail on each of them in this section, but we will discuss some considerations for offline model-based reinforcement learning in Section 5.

2.2 Offline Reinforcement Learning

The offline reinforcement learning problem can be defined as a *data-driven* formulation of the reinforcement learning problem. The end goal is still to optimize the objective in Equation (1). However, the agent no longer has the ability to interact with the environment and collect additional transitions using the behavior policy. Instead, the learning algorithm is provided with a *static* dataset of transitions, $\mathcal{D} = \{(\mathbf{s}^i_t, \mathbf{a}^i_t, \mathbf{s}^i_{t+1}, r^i_t)\}$, and must learn the best policy it can using this dataset. This formulation more closely resembles the standard supervised learning problem statement, and we can regard $\mathcal{D}$ as the *training set* for the policy. In essence, offline reinforcement learning requires the learning algorithm to derive a sufficient understanding of the dynamical system underlying the MDP $\mathcal{M}$ entirely from a fixed dataset, and then construct a policy $\pi(\mathbf{a}|\mathbf{s})$ that attains the largest possible cumulative reward *when it is actually used to interact with the MDP*. We will use π_β to denote the distribution over states and actions in $\mathcal{D}$, such that we assume that the state-action tuples $(\mathbf{s}, \mathbf{a}) \in \mathcal{D}$

are sampled according to $\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$, and the actions are sampled according to the behavior policy, such that $\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})$.

This problem statement has been presented under a number of different names. The term "off-policy reinforcement learning" is typically used as an umbrella term to denote all reinforcement learning algorithms that can employ datasets of transitions $\mathcal{D}$ where the corresponding actions in each transition were collected with any policy *other* than the current policy $\pi(\mathbf{a}|\mathbf{s})$. Q-learning algorithms, actor-critic algorithms that utilize Q-functions, and many model-based reinforcement learning algorithm are off-policy algorithms. However, off-policy algorithms still often employ additional interaction (i.e., online data collection) during the learning process. Therefore, the term "fully off-policy" is sometimes used to indicate that no additional online data collection is performed. Another commonly used term is "batch reinforcement learning" (Ernst et al., 2005; Riedmiller, 2005; Lange et al., 2012). While this term has been used widely in the literature, it can also cause some amount of confusion, since the use of a "batch" in an iterative learning algorithm can also refer to a method that consumes a batch of data, updates a model, and then obtains a different batch, as opposed to a traditional online learning algorithm, which consumes one sample at a time. In fact, Lange et al. (2012) further introduces qualifiers "pure" and "growing" batch reinforcement learning to clarify this. To avoid this confusion, we will instead use the term "offline reinforcement learning" in this tutorial.

The offline reinforcement learning problem can be approached using algorithms from each of the four categories covered in the previous section, and in principle any off-policy RL algorithm *could* be used as an offline RL algorithm. For example, a simple offline RL method can be obtained simply by using Q-learning without additional online exploration, using $\mathcal{D}$ to pre-populate the data buffer. This corresponds to changing the initialization of $\mathcal{D}$ in Algorithm 2, and setting $S = 0$. However, as we will discuss later, not all such methods are effective in the offline setting.

2.3 Example Scenarios

Before delving deeper into the technical questions surrounding offline RL, we will first discuss a few example scenarios where offline RL might be utilized. These scenarios will help us to understand the factors that we must consider when designing offline RL methods that are not only convergent and principled, but also likely to work well in practice. A more complete discussion of actual applications is provided in Section 6.

Decision making in health care. An example health care scenario might formulate a Markov decision process to model the process of diagnosing and treating a patient, where actions correspond to various available interventions (e.g., diagnostic tests and treatments), and observations correspond to the patient's symptoms and results of diagnostic tests. A partially observed MDP formulation may be most suitable in such cases. Conventional active reinforcement learning in such scenarios might be prohibitively dangerous – even utilizing a fully trained policy to treat a patient is a difficult prospect for clinicians, and deploying a partially trained policy would be out of the question. Therefore, offline RL might be the only viable path to apply reinforcement learning in such settings. Offline data would then be obtained from treatment histories of real patients, with the "actions" that were selected by their physician.

Learning goal-directed dialogue policies. Dialogue can be viewed as interactive sequential decision making problem, which can also be modeled as an MDP, particularly when the dialogue is goal-directed (e.g., a chat bot on an e-commerce website that is offering information about a product to persuade a user to make a purchase). However, since the goal for such agents is to interact successfully with real humans, collecting trials requires interacting with live humans, which may be prohibitively expensive at the scale needed to train effective conversational agents. However, offline data can be collected directly from humans, and may indeed be natural to collect in any application domain where the aim is to partially or completely supplant human operators, by recording the interactions that are already taking place with the human-operated system.

Learning robotic manipulation skills. In a robotic manipulation setting, active reinforcement learning may in fact be quite feasible. However, we might want to learn policies for a variety of robotic skills (e.g., all of the steps necessary to prepare a variety of meals for a home cooking robot) that generalize effectively over different environments and settings. In that case, each skill by itself might require a very large amount of interaction, as we would not only need to collect enough data to

learn the skill, but enough data such that this skill generalizes effectively to all the situations (e.g., all the different homes) in which the robot might need to perform it. With offline RL, we could instead imagine including all of the data the robot has ever collected for *all* of its previously learned skills in the data buffer for each new skill that it learns. In this way, some skills could conceivably be acquired without any new data collection, if they can be assembled from parts of previously learned behaviors (e.g., cooking a soup that includes onions and carrots can likely be learned from experience cooking a soup with onions and meat, as well as another soup with carrots and cucumbers). In this way, offline RL can effectively utilize *multi-task* data.

2.4 What Makes Offline Reinforcement Learning Difficult?

Offline reinforcement learning is a difficult problem for multiple reasons, some of which are reasonably clear, and some of which might be a bit less clear. Arguably the most obvious challenge with offline reinforcement learning is that, because the learning algorithm must rely entirely on the static dataset $\mathcal{D}$, there is no possibility of improving exploration: exploration is *outside* the scope of the algorithm, so if $\mathcal{D}$ does not contain transitions that illustrate high-reward regions of the state space, it may be impossible to discover those high-reward regions. However, because there is nothing that we can do to address this challenge, we will not spend any more time on it, and will instead assume that $\mathcal{D}$ adequately covers the space of high-reward transitions to make learning feasible.[2]

A more subtle but practically more important challenge with offline reinforcement learning is that, at its core, offline reinforcement learning is about making and answering counterfactual queries. Counterfactual queries are, intuitively, "what if" questions. Such queries require forming hypotheses about what *might* happen if the agent were to carry out a course of action different from the one seen in the data. This is a necessity in offline RL, since we if we want the learned policy to perform better than the behavior seen in the dataset $\mathcal{D}$, we must execute a sequence of actions that is in some way different. Unfortunately, this strains the capabilities of many of our current machine learning tools, which are designed around the assumption that the data is independent and identically distributed (i.i.d.). That is, in standard supervised learning, the goal is to train a model that attains good performance (e.g., high accuracy) on data coming from the same distribution as the training data. In offline RL, the whole point is to learn a policy that does something *differently* (presumably better) from the pattern of behavior observed in the dataset $\mathcal{D}$.

The fundamental challenge with making such counterfactual queries is distributional shift: while our function approximator (policy, value function, or model) might be trained under one distribution, it will be evaluated on a different distribution, due both to the change in visited states for the new policy and, more subtly, by the act of maximizing the expected return. This latter point is discussed in more detail in Section 4. Distributional shift issues can be addressed in several ways, with the simplest one being to constrain something about the learning process such that the distributional shift is bounded. For example, by constraining how much the learned policy $\pi(\mathbf{a}|\mathbf{s})$ differs from the behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$, we can bound state distributional shift (Kakade and Langford, 2002; Schulman et al., 2015).

In this section, we will provide a short theoretical illustration of how harmful distributional shift can be on the performance of policies in MDPs. In this example, based on Ross et al. (2011), we will assume that we are provided with optimal action labels $\mathbf{a}^\star$ at each state $\mathbf{s} \in \mathcal{D}$. One might expect that, under such a strong assumption, the performance of our learned policy should be at least as good as the policies that we can learn with reinforcement learning without such optimal action labels. The goal in this analysis will be to bound the number of mistakes made by the learned policy $\pi(\mathbf{a}|\mathbf{s})$ based on this labeled dataset, denoted as

$$\ell(\pi) = \mathbb{E}_{p_\pi(\tau)}\left[\sum_{t=0}^{H} \delta(\mathbf{a}_t \neq \mathbf{a}_t^\star)\right].$$

If we train $\pi(\mathbf{a}|\mathbf{s})$ with supervised learning (i.e., standard empirical risk minimization) on this labeled dataset, we have the following result from Ross et al. (2011):

Theorem 2.1 (Behavioral cloning error bound)**.** *If $\pi(\mathbf{a}|\mathbf{s})$ is trained via empirical risk minimization on $\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$ and optimal labels $\mathbf{a}^\star$, and attains generalization error ϵ on $\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$, then $\ell(\pi) \leq C + H^2\epsilon$ is the best possible bound on the expected error of the learned policy.*

[2] It is worth noting that defining this notion formally is itself an open problem, and to the best knowledge of the authors, there are no known non-trivial "sufficiency" conditions on $\mathcal{D}$ that allows us to formulate guarantees that any offline reinforcement learning algorithm will recover an optimal or near-optimal policy.

Proof. The proof follows from Theorem 2.1 from Ross et al. (2011) using the 0-1 loss, and the bound is the best possible bound following the example from Ross and Bagnell (2010). □

Interestingly, if we allow for additional data collection, where we follow the learned policy $\pi(\mathbf{a}|\mathbf{s})$ to gather additional states $\mathbf{s} \sim d^{\pi}(\mathbf{s})$, and then access optimal action labels for these new *on-policy* states, the best possible bound becomes substantially better:

Theorem 2.2 (DAgger error bound)**.** *If $\pi(\mathbf{a}|\mathbf{s})$ is trained via empirical risk minimization on $\mathbf{s} \sim d^{\pi}(\mathbf{s})$ and optimal labels $\mathbf{a}^{\star}$, and attains generalization error ϵ on $\mathbf{s} \sim d^{\pi}(\mathbf{s})$, then $\ell(\pi) \leq C + H\epsilon$ is the best possible bound on the expected error of the learned policy.*

Proof. The proof follows from Theorem 3.2 from Ross et al. (2011). This is the best possible bound, because the probability of a mistake at any time step is at least ϵ, and $\sum_{t=1}^{H} \epsilon = H\epsilon$. □

This means that, even with optimal action labels, we get an error bound that is at best quadratic in the time horizon H in the offline case, but linear in H in the online case. Intuitively, the reason for this gap in performance is that, in the offline case, the learned policy $\pi(\mathbf{a}|\mathbf{s})$ may enter into states that are far outside of its training distribution, since $d^{\pi}(\mathbf{s})$ may be very different from $d^{\pi_\beta}(\mathbf{s})$. In these out-of-distribution states, the generalization error bound ϵ no longer holds, since standard empirical risk minimization makes no guarantees about error when encountering out-of-distribution inputs that were not seen during training. Once the policy enters one of these out-of-distribution states, it will keep making mistakes and may remain out-of-distribution for the remainder of the trial, accumulating $O(H)$ error. Since there is a non-trivial chance of entering an out-of-distribution state at every one of the H time steps, the overall error therefore scales as $O(H^2)$. In the on-policy case, such out-of-distribution states are not an issue. Of course, this example is somewhat orthogonal to the main purpose of this tutorial, which is to study offline reinforcement learning algorithms, rather than offline behavioral cloning methods. However, it should serve as a warning, as it indicates that the challenges of distributional shift are likely to cause considerable harm to any policy trained from an offline dataset if care is not taken to minimize its detrimental effects.

3 Offline Evaluation and Reinforcement Learning via Importance Sampling

In this section, we survey offline reinforcement learning algorithms based on direct estimation of policy return. These methods generally utilize some form of importance sampling to either evaluate the return of a given policy, or to estimate the corresponding policy gradient, corresponding to an offline variant of the policy gradient methods discussed in Section 2.1. The most direct way to utilize this idea is to employ importance sampling to estimate $J(\pi)$ with trajectories sampled from $\pi_\beta(\tau)$. This is known as *off-policy evaluation*. In principle, once we can evaluate $J(\pi)$, we can select the most performant policy. In this section, we review approaches for off-policy evaluation with importance sampling and then discuss how these ideas can be used for offline reinforcement learning.

3.1 Off-Policy Evaluation via Importance Sampling

We can naïvely use importance sampling to derive an unbiased estimator of $J(\pi)$ that relies on off-policy trajectories:

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[\frac{\pi_\theta(\tau)}{\pi_\beta(\tau)} \sum_{t=0}^{H} \gamma^t r(\mathbf{s}, \mathbf{a}) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[\left(\prod_{t=0}^{H} \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)} \right) \sum_{t=0}^{H} \gamma^t r(\mathbf{s}, \mathbf{a}) \right] \approx \sum_{i=1}^{n} w_H^i \sum_{t=0}^{H} \gamma^t r_t^i, \end{aligned} \tag{5}$$

where $w_t^i = \frac{1}{n} \prod_{t'=0}^{t} \frac{\pi_\theta(\mathbf{a}_{t'}^i|\mathbf{s}_{t'}^i)}{\pi_\beta(\mathbf{a}_{t'}^i|\mathbf{s}_{t'}^i)}$ and $\{(\mathbf{s}_0^i, \mathbf{a}_0^i, r_0^i, \mathbf{s}_1^i, \ldots)\}_{i=1}^n$ are n trajectory samples from $\pi_\beta(\tau)$ (Precup, 2000). Unfortunately, such an estimator can have very high variance (potentially unbounded if H is infinite) due to the product of importance weights. Self-normalizing the importance weights (i.e., dividing the weights by $\sum_{i=1}^{n} w_H^i$) results in the *weighted importance sampling* estimator (Precup, 2000), which is biased, but can have much lower variance and is still a strongly consistent estimator.

To improve this estimator, we need to take advantage of the statistical structure of the problem. Because r_t does not depend on $\mathbf{s}_{t'}$ and $\mathbf{a}_{t'}$ for $t' > t$, we can drop the importance weights from future time steps, resulting in the *per-decision* importance sampling estimator (Precup, 2000):

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[\sum_{t=0}^{H} \left(\prod_{t'=0}^{t} \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)} \right) \gamma^t r(\mathbf{s}, \mathbf{a}) \right] \approx \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{H} w_t^i \gamma^t r_t^i.$$

As before, this estimator can have high variance, and we can form a weighted per-decision importance estimator by normalizing the weights. Unfortunately, in many practical problems, the weighted per-decision importance estimator still has too much variance to be effective.

If we have an approximate model that can be used to obtain an approximation to the Q-values for each state-action tuple $(\mathbf{s}_t, \mathbf{a}_t)$, which we denote $\hat{Q}^\pi(\mathbf{s}_t, \mathbf{a}_t)$, we can incorporate it into this estimate. Such an estimate can be obtained, for example, by estimating a model of the MDP transition probability $T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and then solving for the corresponding Q-function, or via any other method for approximating Q-values. We can incorporate these estimates as control variates into the importance sampled estimator to get the best of both:

$$J(\pi_\theta) \approx \sum_{i=1}^{n} \sum_{t=0}^{H} \gamma^t \left(w_t^i \left(r_t^i - \hat{Q}^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right) - w_{t-1}^i \mathbb{E}_{\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s}_t)} \left[\hat{Q}^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}) \right] \right). \tag{6}$$

This is known as the doubly robust estimator (Jiang and Li, 2015; Thomas and Brunskill, 2016) because it is unbiased if either π_β is known or if the model is correct. We can also form a weighted version by normalizing the weights. More sophisticated estimators can be formed by training the model with knowledge of the policy to be evaluated (Farajtabar et al., 2018), and by trading off bias and variance more optimally (Thomas and Brunskill, 2016; Wang et al., 2017).

Beyond consistency or unbiased estimates, we frequently desire a (high-probability) guarantee on the performance of a policy. Thomas et al. (2015) derived confidence bounds based on concentration inequalities specialized to deal with the high variance and potentially large range of the importance weighted estimators. Alternatively, we can construct confidence bounds based on distributional assumptions (e.g., asymptotic normality) (Thomas et al., 2015) or via bootstrapping (Thomas et al., 2015; Hanna et al., 2017) which may be less conservative at the cost of looser guarantees.

Such estimators can also be utilized for policy improvement, by searching over policies with respect to their estimated return. In safety-critical applications of offline RL, we would like to improve over the behavior policy with a guarantee that with high probability our performance is no lower than a bound. Thomas et al. (2015) show that we can search for policies using lower confidence bounds on importance sampling estimators to ensure that the safety constraint is met. Alternatively, we can search over policies in a model of the MDP and bound the error of the estimated model with high probability (Ghavamzadeh et al., 2016; Laroche et al., 2017; Nadjahi et al., 2019).

3.2 The Off-Policy Policy Gradient

Importance sampling can also be used to directly estimate the policy gradient, rather than just obtaining an estimate of the value for a given policy. As discussed in Section 2.1, policy gradient methods aim to optimize $J(\pi)$ by computing estimates of the gradient with respect to the policy parameters. We can estimate the gradient with Monte Carlo samples, as in Equation (2), but this requires *on-policy* trajectories (i.e., $\tau \sim \pi_\theta(\tau)$). Here, we extend this approach to the offline setting.

Previous work has generally focused on the *off-policy* setting, where trajectories are sampled from a distinct behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s}) \neq \pi(\mathbf{a}|\mathbf{s})$. However, in contrast to the offline setting, such methods assume we can continually sample new trajectories from π_β, while old trajectories are reused for efficiency. We begin with the off-policy setting, and then discuss additional challenges in extending such methods to the offline setting.

Noting the similar structure between $J(\pi)$ and the policy gradient, we can adapt the techniques for estimating $J(\pi)$ off-policy to the policy gradient

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\beta(\tau)}\left[\frac{\pi_\theta(\tau)}{\pi_\beta(\tau)}\sum_{t=0}^{H}\gamma^t\nabla_\theta\log\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}(\mathbf{s}_t,\mathbf{a}_t)\right]\\ &= \mathbb{E}_{\tau\sim\pi_\beta(\tau)}\left[\left(\prod_{t=0}^{H}\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)}\right)\sum_{t=0}^{H}\gamma^t\nabla_\theta\log\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}(\mathbf{s}_t,\mathbf{a}_t)\right]\\ &\approx \sum_{i=1}^{n}w_H^i\sum_{t=0}^{H}\gamma^t\nabla_\theta\log\pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\hat{A}(\mathbf{s}_t^i,\mathbf{a}_t^i),\end{aligned}$$

where $\{(\mathbf{s}_0^i,\mathbf{a}_0^i,r_0^i,\mathbf{s}_1^i,\ldots)\}_{i=1}^n$ are n trajectory samples from $\pi_\beta(\tau)$ (Precup, 2000; Precup et al., 2001; Peshkin and Shelton, 2002). Similarly, we can self-normalize the importance weights resulting in the *weighted importance sampling* policy gradient estimator (Precup, 2000), which is biased, but can have much lower variance and is still a consistent estimator.

If we use the Monte Carlo estimator with baseline for $\hat{A}$ (i.e., $\hat{A}(\mathbf{s}_t^i,\mathbf{a}_t^i)=\sum_{t'=t}^{H}\gamma^{t'-t}r_{t'}-b(\mathbf{s}_t^i)$), then because r_t does not depend on $\mathbf{s}_{t'}$ and $\mathbf{a}_{t'}$ for $t'>t$, we can drop importance weights in the future, resulting in the *per-decision* importance sampling policy gradient estimator (Precup, 2000):

$$\nabla_\theta J(\pi_\theta)\approx\sum_{i=1}^{n}\sum_{t=0}^{H}w_t^i\gamma^t\left(\sum_{t'=t}^{H}\gamma^{t'-t}\frac{w_{t'}^i}{w_t^i}r_{t'}-b(\mathbf{s}_t^i)\right)\nabla_\theta\log\pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i).$$

As before, this estimator can have high variance, and we can form a weighted per-decision importance estimator by normalizing the weights. Paralleling the development of doubly robust estimators for policy evaluation, doubly robust estimators for the policy gradient have also been derived (Gu et al., 2017a; Huang and Jiang, 2019; Pankov, 2018; Cheng et al., 2019). Unfortunately, in many practical problems, these estimators have too high variance to be effective.

Practical off-policy algorithms derived from such estimators can also employ regularization, such that the learned policy $\pi_\theta(\mathbf{a}|\mathbf{s})$ does not deviate too far from the behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$, thus keeping the variance of the importance weights from becoming too large. One example of such a regularizer is the *soft max* over the (unnormalized) importance weights (Levine and Koltun, 2013). This regularized gradient estimator $\nabla_\theta\bar{J}(\pi_\theta)$ has the following form:

$$\nabla_\theta\bar{J}(\pi_\theta)\approx\left(\sum_{i=1}^{n}w_H^i\sum_{t=0}^{H}\gamma^t\nabla_\theta\log\pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\hat{A}(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)+\lambda\log\left(\sum_{i=1}^{n}w_H^i\right).$$

It is easy to check that $\sum_i w_H^i\to 1$ as $n\to\infty$, meaning that this gradient estimator is consistent. However, with a finite number of samples, such an estimator automatically adjusts the policy π_θ to ensure that at least one sample has a high (unnormalized) importance weight. More recent deep reinforcement learning algorithms based on importance sampling often employ a sample-based KL-divergence regularizer (Schulman et al., 2017), which has a functional form mathematically similar to this one when also utilizing an entropy regularizer on the policy π_θ.

3.3 Approximate Off-Policy Policy Gradients

The importance-weighted policy objective requires multiplying per-action importance weights over the time steps, which leads to very high variance. We can derive an approximate importance-sampled gradient by using the state distribution of the behavior policy, $d^{\pi_\beta}(\mathbf{s})$, in place of that of the current policy, $d^\pi(\mathbf{s})$. This results in a biased gradient due to the mismatch in state distributions, but can provide reasonable learning performance in practice. The corresponding objective, which we will denote $J_{\pi_\beta}(\pi_\theta)$ to emphasize its dependence on the behavior policy's state distribution, is given by

$$J_{\pi_\beta}(\pi_\theta)=\mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s})}\left[V^\pi(\mathbf{s})\right].$$

Note that $J_{\pi_\beta}(\pi_\theta)$ and $J(\pi_\theta)$ differ in the distribution of states under which the return is estimated ($d^{\pi_\beta}(\mathbf{s})$ vs. $d^\pi(\mathbf{s})$), making $J_{\pi_\beta}(\pi_\theta)$ a biased estimator for $J(\pi_\theta)$. This may lead to suboptimal solutions in certain cases (see Imani et al. (2018) for some examples). However, expectations under

state distributions from $d^{\pi_\beta}(\mathbf{s})$ can be calculated easily by sampling states from the dataset $\mathcal{D}$ in the offline case, removing the need for importance sampling.

Recent empirical work has found that this bias is acceptable in practice (Fu et al., 2019). Differentiating the objective and applying a further approximation results in the *off-policy policy gradient* (Degris et al., 2012):

$$\begin{aligned}\nabla_\theta J_{\pi_\beta}(\pi_\theta) &= \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[Q^{\pi_\theta}(\mathbf{s},\mathbf{a})\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) + \nabla_\theta Q^{\pi_\theta}(\mathbf{s},\mathbf{a})\right] \\ &\approx \mathbb{E}_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[Q^{\pi_\theta}(\mathbf{s},\mathbf{a})\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})\right].\end{aligned}$$

Degris et al. (2012) show that under restrictive conditions, the approximate gradient preserves the local optima of $J_{\pi_\beta}(\pi)$. This approximate gradient is used as a starting point in many widely used deep reinforcement learning algorithms (Silver et al., 2014; Lillicrap et al., 2015; Wang et al., 2016; Gu et al., 2017b).

To compute an estimate of the approximate gradient, we additionally need to estimate $Q^{\pi_\theta}(\mathbf{s},\mathbf{a})$ from the off-policy trajectories. Basic methods for doing this were discussed in Section 2.1, and we defer a more in-depth discussion of offline state-action value function estimators to Section 4. Some estimators use action samples, which required an importance weight to correct from π_β samples to π_θ samples. Further improvements can be obtained by introducing control variates and clipping importance weights to control variance (Wang et al., 2016; Espeholt et al., 2018).

3.4 Marginalized Importance Sampling

If we would like to avoid the bias incurred by using the off-policy state distribution and the high variance from using per-action importance weighting, we can instead attempt to directly estimate the *state-marginal importance ratio* $\rho^{\pi_\theta}(\mathbf{s}) = \frac{d^{\pi_\theta}(\mathbf{s})}{d^{\pi_\beta}(\mathbf{s})}$. An estimator using state marginal importance weights can be shown to be have no greater variance than using the product of per-action importance weights. However, computing this ratio exactly is generally intractable. Only recently have practical methods for estimating the marginalized importance ratio been introduced (Sutton et al., 2016; Yu, 2015; Hallak et al., 2015, 2016; Hallak and Mannor, 2017; Nachum et al., 2019a; Zhang et al., 2020a; Nachum et al., 2019b). We discuss some key aspects of these methods next.

Most of these methods utilize some form of dynamic programming to estimate the importance ratio ρ^π. Based on the form of the underlying Bellman equation used, we can classify them into two categories: methods that use a "**forward**" Bellman equation to estimate the importance ratios directly, and methods that use a "**backward**" Bellman equation on a functional that resembles a value function and then derive the importance ratios from the learned value functional.

Forward Bellman equation based approaches. The state-marginal importance ratio, $\rho^\pi(\mathbf{s})$, satisfies a kind of "forward" Bellman equation:

$$\forall \mathbf{s}', \quad \underbrace{d^{\pi_\beta}(\mathbf{s}')\rho^\pi(\mathbf{s}')}_{:=(d^{\pi_\beta}\circ\rho^\pi)(\mathbf{s}')} = \underbrace{(1-\gamma)d_0(\mathbf{s}') + \gamma\sum_{\mathbf{s},\mathbf{a}} d^{\pi_\beta}(\mathbf{s})\rho^\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})T(\mathbf{s}'|\mathbf{s},\mathbf{a})}_{:=(\bar{\mathcal{B}}^\pi\circ\rho^\pi)(\mathbf{s}')}. \tag{7}$$

This relationship can be leveraged to perform temporal difference updates to estimate the state-marginal importance ratio under the policy.

For example, when stochastic approximation is used, Gelada and Bellemare (2019) use the following update rule in order to estimate $\rho^\pi(\mathbf{s}')$ online:

$$\hat{\rho}^\pi(\mathbf{s}') \leftarrow \hat{\rho}^\pi(\mathbf{s}') + \alpha\left[(1-\gamma) + \gamma\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\hat{\rho}^\pi(\mathbf{s}) - \hat{\rho}^\pi(\mathbf{s}')\right], \tag{8}$$

with $\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}), \mathbf{a}\sim\pi_\beta(\mathbf{a}|\mathbf{s}), \mathbf{s}'\sim T(\mathbf{s}'|\mathbf{s},\mathbf{a})$. Several additional techniques, including using TD(λ) estimates and automatic adjustment of feature dimensions, have been used to stabilize learning. We refer the readers to Hallak and Mannor (2017) and Gelada and Bellemare (2019) for a detailed discussion. Gelada and Bellemare (2019) also discusses several practical tricks, such as soft-normalization and discounted evaluation, to adapt these methods to deep Q-learning settings, unlike prior work that mainly focuses on linear function approximation. Wen et al. (2020) view to problem from the lens of power iteration, and propose a variational power method approach to combine function approximation

and power iteration to estimate ρ^π. Imani et al. (2018); Zhang et al. (2019) show that similar methods can be used to estimate the off-policy policy gradient and thus be used in an off-policy actor critic method.

Alternatively, Liu et al. (2018) propose to use an adversarial approach to obtain the state-marginal importance ratios. From Eq. 7, they derive a functional

$$L(\rho, f) = \gamma \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}} \left[\left(\rho(\mathbf{s}) \frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} - \rho(\mathbf{s}') \right) f(\mathbf{s}') \right] + (1-\gamma) \mathbb{E}_{\mathbf{s}_0 \sim d_0}[(1-\rho(\mathbf{s}))f(\mathbf{s})] \quad (9)$$

such that $L(\rho, f) = 0, \forall\, f$ if and only if $\rho = \rho^\pi$. Therefore, we can learn ρ by minimizing a worst-case estimate of $L(\rho, f)^2$, by solving an adversarial, saddle-point optimization: $\min_\rho \max_f L(\rho, f)^2$. Recent work (Mousavi et al., 2020; Kallus and Uehara, 2019a,b; Tang et al., 2019; Uehara and Jiang, 2019) has refined this approach, in particular removing the need to have access to π_β. Once ρ^* is obtained, Liu et al. (2019) use this estimator to compute the off-policy policy gradient.

Zhang et al. (2020a) present another off-policy evaluation method that computes the importance ratio for the state-action marginal, $\rho^\pi(\mathbf{s}, \mathbf{a}) := \frac{d^\pi(\mathbf{s},\mathbf{a})}{d^{\pi_\beta}(\mathbf{s},\mathbf{a})}$, by directly optimizing a variant of the Bellman residual error corresponding to a modified forward Bellman equation, that includes actions, shown in Equation 10.

$$\underbrace{d^{\pi_\beta}(\mathbf{s}', \mathbf{a}')\rho^\pi(\mathbf{s}', \mathbf{a}')}_{:=(d^{\pi_\beta} \circ \rho^\pi)(\mathbf{s}', \mathbf{a}')} = \underbrace{(1-\gamma) d_0(\mathbf{s}')\pi(\mathbf{a}'|\mathbf{s}') + \gamma \sum_{\mathbf{s},\mathbf{a}} d^{\pi_\beta}(\mathbf{s}, \mathbf{a}) \rho^\pi(\mathbf{s}, \mathbf{a}) \pi(\mathbf{a}|\mathbf{s}) T(\mathbf{s}'|\mathbf{s}, \mathbf{a})}_{:=(\bar{\mathcal{B}}^\pi \circ \rho^\pi)(\mathbf{s}', \mathbf{a}')}. \quad (10)$$

Their approach can be derived by applying a divergence metric (such as an f-divergence, which we will review in Section 4) between the two sides of the modified forward Bellman equation in Equation 10, while additionally constraining the importance ratio, $\rho^\pi(\mathbf{s}, \mathbf{a})$, to integrate to 1 in expectation over the dataset, $\mathcal{D}$, to prevent degenerate solutions, as follows

$$\min_{\rho^\pi} \; D_f \left(\left(\bar{\mathcal{B}}^\pi \circ \rho^\pi \right)(\mathbf{s}, \mathbf{a}), \left(d^{\pi_\beta} \circ \rho^\pi \right)(\mathbf{s}, \mathbf{a}) \right) \quad \text{s.t.} \quad \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}}[\rho^\pi(\mathbf{s}, \mathbf{a})] = 1. \quad (11)$$

They further apply tricks inspired from dual embeddings (Dai et al., 2016) to make the objective tractable, and to avoid the bias caused due to sampled estimates. We refer the readers to Zhang et al. (2020a) for further discussion.

Zhang et al. (2020b) show that primal-dual solvers may not be able to solve Eqn. 11, and modify the objective in by replacing the f-divergence with a norm induced by $1/d^{\pi_\beta}$. This creates an optimization problem that is provably convergent under linear function approximation.

$$\min_{\rho^\pi} \; \frac{1}{2} || \left(\left(\bar{\mathcal{B}}^\pi \circ \rho^\pi \right)(\mathbf{s}, \mathbf{a}) - \left(d^{\pi_\beta} \circ \rho^\pi \right)(\mathbf{s}, \mathbf{a}) \right| |^2_{(d^{\pi_\beta})^{-1}} + \frac{\lambda}{2} (\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}}[\rho^\pi(\mathbf{s}, \mathbf{a})] - 1)^2. \quad (12)$$

Backward Bellman equation based approaches via convex duality. Finally, we discuss methods for off-policy evaluation and improvement that utilize a backward Bellman equation – giving rise to a value-function like functional – by exploiting convex duality. Because these methods start from an optimization perspective, they can bring to bear the mature tools of convex optimization and online learning. Lee and He (2018) extend a line of work applying to convex optimization techniques to policy optimization (Chen and Wang, 2016; Wang and Chen, 2016; Dai et al., 2017a,b) to the off-policy setting. They prove sample complexity bounds in the off-policy setting, however, extending these results to practical deep RL settings has proven challenging.

Nachum et al. (2019a) develop similar ideas for off-policy evaluation. The key idea is to devise a convex optimization problem with ρ^π as its optimal solution. The chosen optimization problem is

$$\rho^\pi = \arg \min_{x: \mathcal{S} \times \mathcal{A} \to \mathbb{R}} \; \frac{1}{2} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}} \left[x(\mathbf{s}, \mathbf{a})^2 \right] - \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s}), \mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[x(\mathbf{s}, \mathbf{a}) \right]. \quad (13)$$

Unfortunately, this objective requires samples from the on-policystate-marginal distribution, $d^\pi(\mathbf{s})$. The key insight is a change of variables, $x(s, a) = \nu(\mathbf{s}, \mathbf{a}) - \mathbb{E}_{\mathbf{s}' \sim T(\mathbf{s}'|\mathbf{s},\mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}[\nu(\mathbf{s}', \mathbf{a}')]$ and introduce the variable $\nu(\mathbf{s}, \mathbf{a})$ to simplify the expression in Equation 13, and obtain the "backward" dynamic programming procedure shown in Equation 14. For brevity, we define a modified Bellman

operator, $\tilde{\mathcal{B}}^\pi \nu(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{\mathbf{s}' \sim T(\mathbf{s}'|\mathbf{s},\mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}[\nu(\mathbf{s}', \mathbf{a}')]$, that is similar to the expression for $\mathcal{B}^\pi$ without the reward term $r(s, a)$.

$$\min_{\nu: \mathcal{S} \times \mathcal{A} \to \mathbb{R}} \frac{1}{2} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}} \left[\left(\nu(\mathbf{s}, \mathbf{a}) - \tilde{\mathcal{B}}^\pi \nu(\mathbf{s}, \mathbf{a}) \right)^2 \right] - \mathbb{E}_{\mathbf{s}_0 \sim d_0(\mathbf{s}_0), \mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s}_0)} \left[\nu(\mathbf{s}_0, \mathbf{a}) \right]. \tag{14}$$

Remarkably, Equation 14 does not require on-policy samples to evaluate. Given an optimal solution for the objective in Equation 14, denoted as ν^*, we can obtain the density ratio, ρ^π, using the relation, $\rho^\pi(\mathbf{s}, \mathbf{a}) = \nu^*(\mathbf{s}, \mathbf{a}) - \tilde{\mathcal{B}}^\pi \nu^*(\mathbf{s}, \mathbf{a})$. The density ratio can then be used for off-policy evaluation and improvement.

Nachum et al. (2019b); Nachum and Dai (2020) build on a similar framework to devise an off-policy RL algorithm. The key idea is to obtain an estimate of the on-policy policy gradient for a state-marginal *regularized* RL objective by solving an optimization problem. The regularizer applied in this family of methods is the f-divergence between the state(-action) marginal of the learned policy and the state-action marginal of the dataset. We will cover f-divergences in detail in Section 4. The f-divergence regularized RL problem, with a tradeoff factor, α, is given by:

$$\max_\pi \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s}), \mathbf{a} \sim \pi(\cdot|\mathbf{s})} \left[r(\mathbf{s}, \mathbf{a}) \right] - \alpha D_f(d^\pi(\mathbf{s}, \mathbf{a}), d^{\pi_\beta}(\mathbf{s}, \mathbf{a})). \tag{15}$$

By exploiting the variational (dual) form of the f-divergence shown below

$$D_f(p, q) = \max_{x: \mathcal{S} \times \mathcal{A} \to \mathbb{R}} \left(\mathbb{E}_{\mathbf{y} \sim p(\mathbf{y})}[x(\mathbf{y})] - \mathbb{E}_{\mathbf{y} \sim q(\mathbf{y})}[f^*(x(\mathbf{y}))] \right), \tag{16}$$

and then applying a change of variables from x to Q (c.f., (Nachum et al., 2019a)) where Q satisfies $Q(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim T(\mathbf{s},\mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')} \left[r(\mathbf{s}, \mathbf{a}) - \alpha x(\mathbf{s}, \mathbf{a}) + \gamma Q(\mathbf{s}', \mathbf{a}') \right]$, we obtain a saddle-point optimization problem for optimizing the regularized RL objective,

$$\max_\pi \min_Q L(Q, \pi_\beta, \pi) \coloneqq \mathbb{E}_{\mathbf{s}_0 \sim d_0(\mathbf{s}_0), \mathbf{a} \sim \pi(\cdot|\mathbf{s}_0)} \left[Q(\mathbf{s}_0, \mathbf{a}) \right] + \alpha \mathbb{E}_{\mathbf{s},\mathbf{a} \sim d^{\pi_\beta}(\mathbf{s},\mathbf{a})} \left[f^* \left(\frac{r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim T(\mathbf{s},\mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}[Q(\mathbf{s}', \mathbf{a}')] - Q(\mathbf{s}, \mathbf{a})}{\alpha} \right) \right].$$

When $f(x) = x^2$, $f^*(x) = x^2$ and this objective reduces to applying a regular actor-critic algorithm as discussed in Section 2.1 along with an additional term minimizing Q-values at the initial state $\mathbf{s}_0$.

It can also be shown that the derivative with respect to the policy of $L(Q^*, \pi_\beta, \pi)$, at the optimal Q-function, Q^*, is precisely equal to the on-policy policy gradient in the regularized policy gradient problem:

$$\frac{\partial}{\partial \pi} L(Q^*, \pi_\beta, \pi) = \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s}), \mathbf{a} \sim \pi(\cdot|\mathbf{s})} \left[\tilde{Q}^\pi(\mathbf{s}, \mathbf{a}) \cdot \nabla_\pi \log \pi(\mathbf{a}|\mathbf{s}) \right], \tag{17}$$

where $\tilde{Q}^\pi$ is the action-value function corresponding to the regularized RL problem.

Finally, we note that this is a rapidly developing area and recent results suggest that many of these methods can be unified under a single framework (Tang et al., 2019; Nachum and Dai, 2020).

3.5 Challenges and Open Problems

The methods discussed in this section utilize some form of importance sampling to either estimate the return of the current policy π_θ, or estimate the gradient of this return. The policy improvement methods discussed in this section have been developed primarily for a classic off-policy learning setting, where additional data is collected online, but previously collected data is reused to improve efficiency. To the best of our knowledge, such methods have not generally been applied in the offline setting, with the exception of the evaluation and high-confidence improvement techniques in Section 3.1.

Applying such methods in the fully offline setting poses a number of major challenges. First, importance sampling already suffer from high variance, and this variance increases dramatically in the sequential setting, since the importance weights at successive time steps are multiplied together (see, e.g., Equation (5)), resulting in exponential blowup. Approximate and marginalized importance sampling methods alleviate this issue to some degree by avoiding multiplication of importance weights over multiple time steps, but the fundamental issue still remains: when the

behavior policy π_β is too different from the current learned policy π_θ, the importance weights will become degenerate, and any estimate of the return or the gradient will have too much variance to be usable, especially in high-dimensional state and action spaces, or for long-horizon problems. For this reason, importance-sampled estimators are most suitable in the case where the policy only deviates by a limited amount from the behavior policy. In the classic off-policy setting, this is generally the case, since new trajectories are repeatedly collected and added to the dataset using the latest policy, but in the offline setting this is generally not the case. Thus, the maximum improvement that can be reliably obtained via importance sampling is limited by (i) the suboptimality of the behavior policy; (ii) the dimensionality of the state and action space; (iii) the effective horizon of the task. The second challenge is that the most effective of these off-policy policy gradient methods either requires estimating the value function, or the state-marginal density ratios via dynamic programming. As several prior works have shown, and as we will review in Section 4, dynamic programming methods suffer from issues pertaining to out-of-distribution queries in completely offline settings, making it hard to stably learn the value function without additional corrections. This problem is not as severe in the classic off-policy setting, which allows additional data collection. Nonetheless, as we will discussion in Section 6, a number of prior applications have effectively utilized importance sampling in an offline setting.

4 Offline Reinforcement Learning via Dynamic Programming

Dynamic programming methods, such as Q-learning algorithms, in principle can offer a more attractive option for offline reinforcement learning as compared to pure policy gradients. As discussed in Section 2.1, dynamic programming methods aim to learn a state or state-action value function, and then either use this value function to directly recover the optimal policy or, as in the case of actor-critic methods, use this value function to estimate a gradient for the expected returns of a policy. Basic offline dynamic programming algorithms can be constructed on the basis of fitted Q-learning methods (Ernst et al., 2005; Riedmiller, 2005; Hafner and Riedmiller, 2011), as well as policy iteration methods (Sutton and Barto, 1998). The generic Q-learning and actor-critic algorithms presented in Algorithm 2 and Algorithm 3 in Section 2.1 can in principle be utilized as offline reinforcement learning, simply by setting the number of collection steps S to zero, and initializing the buffer to be non-empty. Such algorithms can be viable for offline RL, and indeed have been used as such even in recent deep reinforcement learning methods. We will discuss an older class of sucm methods, based on linear function approximation, in Section 4.1, but such techniques have also been used effectively with deep neural network function approximators. For example, Kalashnikov et al. (2018) describes a Q-learning algorithm called QT-Opt that was able to learn effective vision-based robotic grasping strategies from about 500,000 grasping trials logged over the course of previous experiments, but observes that additional online fine-tuning still improved the performance of the policy considerably over the one trained purely from logged data. Some prior works on offline RL have also noted that, for some types of datasets, conventional dynamic programming algorithms, such as deep Q-learning or deterministic actor-critic algorithms, can in fact work reasonably well (Agarwal et al., 2019).

However, as we will discuss in Section 4.2, such methods suffer from a number of issues when all online collection is halted, and only offline data is used. These issues essentially amount to distributional shift, as discussed in Section 2.4. Solutions to this issue can be broadly grouped into two categories: **policy constraint** methods, discussed in Section 4.3, which constrain the learned policy π to lie close to the behavior policy π_β, thus mitigating distributional shift, and **uncertainty-based** methods, discussed in Section 4.4, which attempt to estimate the epistemic uncertainty of Q-values, and then utilize this uncertainty to detect distributional shift. We will discuss both classes of distributional shift corrections, and then conclude with perspectives on remaining challenges and open problems in Section 4.6.

4.1 Off-Policy Value Function Estimation with Linear Value Functions

We first discuss standard offline reinforcement learning methods based on value function and policy estimation with linear function approximators, which do not by themselves provide any mitigate for distributional shift, but can work well when good linear features are available. While modern deep reinforcement learning methods generally eschew linear features in favor of non-linear neural network function approximators, linear methods constitute an important class of offline reinforcement learning algorithms in the literature (Lange et al., 2012; Lagoudakis and Parr, 2003). We begin

with algorithms that use a linear function to approximate the Q-function, such that $Q_\phi \approx \mathbf{f}(s,a)^T\phi$, where $\mathbf{f}(s,a) \in \mathbb{R}^d$ is a feature vector associated with a state-action pair. This parametric Q-function can then be estimated for a given policy $\pi(\mathbf{a}|\mathbf{s})$ using data from a different behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$, with state visitation frequency $d^{\pi_\beta}(\mathbf{s})$. As discussed in Section 2.1, the Q-function Q^π for a given policy $\pi(\mathbf{a}|\mathbf{s})$ must satisfy the Bellman equation, given in full in Equation (3), and written in Bellman operator notation as $\vec{Q^\pi} = \mathcal{B}^\pi\vec{Q^\pi}$.

When linear function approximation is used to represent the Q-function, the Q-function for a policy can be represented as the solution of a linear system, and hence can be computed via least squares minimization, since the Bellman operator $\mathcal{B}^\pi$ is linear. This provides a convenient way to compute Q^π directly in closed form, as compared to using gradient descent in Algorithm 3. The resulting Q-value estimates can then be used to improve the policy. We start with a discussion of different ways of solving the linear system for computing Q^π. To recap, the Q-function is a linear function on a feature vector $\mathbf{f}(\mathbf{s},\mathbf{a})$, which we can express in tabular form as $\mathbf{F} \in \mathbb{R}^{|S||A|\times d}$, such that $\vec{Q}_\phi = \mathbf{F}\phi$. Multiple procedures that can be used to obtain a closed form expression for the optimal ϕ for a given policy π and, as discussed in by Sutton et al. (2009) and Lagoudakis and Parr (2003), these procedures may each yield different solutions under function approximation. We first summarize two methods for selecting ϕ to approximate Q^π for a given policy π, and then discuss how to utilize these methods in a complete reinforcement learning method.

Bellman residual minimization. The first approach selects ϕ such that the resulting linear Q-function satisfies the Bellman equation as closely as possible in terms of squared error, which can be obtained via the least squares solution. To derive the corresponding least squares problem, we first write the Bellman equation in terms of the Bellman operator, and expand it using the vectorized expression for the reward function, $\vec{R}$, and a linear operator representing the dynamics and policy backup, which we denote as P^π, such that $(P^\pi\vec{Q})(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'\sim T(\mathbf{s}'|\mathbf{s},\mathbf{a}),\mathbf{a}'\sim\pi(\mathbf{a}'|\mathbf{s}')}[Q(\mathbf{s}',\mathbf{a}')]$. The corresponding expression of the Bellman equation is given by

$$\mathbf{F}\phi \approx \mathcal{B}^\pi\mathbf{F}\phi = \vec{R} + \gamma P^\pi\mathbf{F}\phi \implies (\mathbf{F} - \gamma P^\pi\mathbf{F})\,\phi \approx \vec{R}.$$

Writing out the least squares solution using the normal equations, we obtain

$$\phi = \left((\mathbf{F} - \gamma P^\pi\mathbf{F})^T(\mathbf{F} - \gamma P^\pi\mathbf{F})\right)^{-1}(\mathbf{F} - \gamma P^\pi\mathbf{F})^T\vec{R}.$$

This expression minimizes the ℓ_2 norm of the Bellman residual (the squared difference between the left-hand side and right-hand side of the Bellman equation), and is referred to as the Bellman residual minimizing solution.

Least-squares fixed point approximation. An alternate approach is use projected fixed-point iteration, rather than direct minimization of the Bellman error, which gives rise to the least-squares fixed point approximation. In this approach, instead of minimizing the squared difference between the left-hand side and right-hand side of the Bellman equation, we instead attempt to iterate the Bellman operator to convergence. In the tabular case, as discussed in Section 2.1, we know that iterating $\vec{Q}_{k+1} \leftarrow \mathcal{B}^\pi\vec{Q}_k$ converges, as $k \to \infty$, to $\vec{Q}^\pi$. In the case where we use function approximation to represent $\vec{Q}_k$, we cannot set $\vec{Q}_{k+1}$ to $\mathcal{B}^\pi\vec{Q}_k$ precisely, because there may not be any value of the weights ϕ that represent $\mathcal{B}^\pi\vec{Q}_k$ exactly. We therefore must employ a *projected* fixed point iteration method, where each iterate $\mathcal{B}^\pi\vec{Q}_k = \mathcal{B}^\pi\mathbf{F}\phi_k$ is projected onto the span of $\mathbf{F}$ to obtain ϕ_{k+1}. We can express this projection via a projection operator, Π, such that the projected Bellman iteration is given by $\vec{Q}_{k+1} = \Pi\mathcal{B}^\pi\vec{Q}_k$. We can obtain a solution for this operator by solving the normal equation, and observe that $\Pi = \mathbf{F}(\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T$. We can expand out the projected Bellman iteration expression as:

$$\begin{aligned}
\vec{Q}_{k+1} &= \mathbf{F}(\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T(\vec{R} + \gamma P^\pi\vec{Q}_k) \\
\mathbf{F}\phi_{k+1} &= \mathbf{F}(\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T(\vec{R} + \gamma P^\pi\mathbf{F}\phi_k) \\
\phi_{k+1} &= (\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T(\vec{R} + \gamma P^\pi\mathbf{F}\phi_k). \qquad (18)
\end{aligned}$$

By repeatedly applying this recurrence, we can obtain the fixed point of the projected Bellman operator as $k \to \infty$ (Sutton et al., 2009).

Both methods have been used in the literature, and there is no clear consensus on which approach is preferred, though they yield different solutions in general when the true Q-function Q^π is not

in the span of $\mathbf{F}$ (i.e., cannot be represented by any parameter vector ϕ) (Lagoudakis and Parr, 2003). We might at first surmise that a good linear fitting method should find the Q-function $\mathbf{F}\phi$ that corresponds to a least-squares projection of the true $\vec{Q}^\pi$ onto the hyperplane defined by $\mathbf{F}$. Unfortunately, *neither* the Bellman residual minimization method nor the least-squares fixed point method in general obtains this solution. The Bellman residual minimization does not in general produce a fixed point of *either* the Bellman operator or the projected Bellman operator. However, the solution obtained via Bellman residual minimization may be closer to the true Q-function in terms of ℓ_2 distance, since it is explicitly obtained by minimizing Bellman residual error. Least-squares fixed point iteration obtains a Q-function that is a fixed point of the projected Bellman operator, but may be arbitrarily suboptimal. However, least-squares fixed point iteration can learn solutions that lead, empirically, to better-performing policies (Sutton et al., 2009; Lagoudakis and Parr, 2003). In general, there are no theoretical arguments that justify the superiority of one approach over the other. In practice, least-squares fixed-point iteration can give rise to more effective policies as compared to the Bellman residual, while the Bellman residual minimization approach can be more stable and predictable (Lagoudakis and Parr, 2003).

Least squares temporal difference Q-learning (LSTD-Q). Now that we have covered different approaches to solve a linear system of equations to obtain an approximation to Q^π, we describe LSTD-Q, a method for estimating Q^π from a static dataset, directly from samples. This method incrementally estimates the terms $\mathbf{F}^T(\mathbf{F} - \gamma P^\pi \mathbf{F})$ and $\mathbf{F}^T\vec{R}$, which appear in Equation (18), and then inverts this sampled estimate to then obtain ϕ. We defer further discussion on LSTD-Q to Lagoudakis and Parr (2003), which also describes several computationally efficient implementations of the LSTD-Q algorithm. Note that the LSTD-Q algorithm is not directly applicable to estimating Q^*, the optimal Q-function, since the optimal Bellman equation for Q^* is not linear due to the maximization, and thus cannot be solved in closed form.

Least squares policy iteration (LSPI). Finally, we discuss least-squares policy iteration (LSPI), a classical offline reinforcement learning algorithm that performs approximate policy iteration (see discussion in Section 2.1) using a linear approximation to the Q-function. LSPI uses LSTD-Q as an intermediate sub-routine to perform approximate policy evaluation, thereby obtaining an estimate for Q^π, denoted Q_ϕ. Then, the algorithm sets the next policy iterate to be equal to the greedy policy corresponding to the approximate Q_ϕ, such that $\pi_{k+1}(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}))$. An important and useful characteristic of LSPI is that it does not require a separate approximate representation for the policy when the actions are discrete, and hence removes any source of error that arises due to function approximation in the actor in actor-critic methods. However, when the action space is continuous, a policy gradient method similar to the generic actor-critic algorithm in Algorithm 3 can be used to optimize a parametric policy.

4.2 Distributional Shift in Offline Reinforcement Learning via Dynamic Programming

Both the linear and non-linear dynamic programming methods discussed so far, in Section 2.1 and Section 4.1 above, can *in principle* learn from offline data, collected according to a different (unknown) behavioral policy $\pi_\beta(\mathbf{a}|\mathbf{s})$, with state visitation frequency $d^{\pi_\beta}(\mathbf{s})$. However, in practice, these procedures can result in very poor performance, due to the distributional shift issues alluded to in Section 2.4.

Distributional shift affects offline reinforcement learning via dynamic programming both at test time and at training time. At test time, the state visitation frequency $d^\pi(\mathbf{s})$ induced by a policy trained with offline RL will differ systematically from the state visitation frequency of the training data, $d^{\pi_\beta}(\mathbf{s})$. This means that, as in the case of policy gradients, a policy trained via an actor-critic method may produce unexpected and erroneous actions in out-of-distribution states $\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$, as can the implicit greedy policy induced by a Q-function in a Q-learning method. One way to mitigate this distributional shift is to limit how much the learned policy can diverge from the behavior policy. For example, by constraining $\pi(\mathbf{a}|\mathbf{s})$ such that $D_{\text{KL}}(\pi(\mathbf{a}|\mathbf{s})\|\pi_\beta(\mathbf{a}|\mathbf{s})) \leq \epsilon$, we can bound $D_{\text{KL}}(d^\pi(\mathbf{s})\|d^{\pi_\beta}(\mathbf{s}))$ by δ, which is $O(\epsilon/(1-\gamma)^2)$ (Schulman et al., 2015). This bound is very loose in practice, but nonetheless suggests that the effects of state distribution shift can potentially be mitigated by bounding how much the learned policy can deviate from the behavior policy that collected the offline training data. This may come at a substantial cost in final performance however, as the behavior policy – and any policy that is close to it – may be much worse than the best policy that can be learned from the offline data.

It should be noted that, for the algorithms discussed previously, state distribution shift affects test-time performance, but has no effect on training, since neither the policy nor the Q-function is ever evaluated at any state that was not sampled from $d^{\pi_\beta}(\mathbf{s})$. However, the training process *is* affected by *action* distribution shift, because the target values for the Bellman backups in Equation (3) depend on $\mathbf{a}_{t+1} \sim \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$. While this source of distribution shift may at first seem insignificant, it in fact presents one of the largest obstacles for practical application of approximate dynamic programming methods to offline reinforcement learning. Since computing the target values in Equation (3) requires evaluating $Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$, where $\mathbf{a}_{t+1} \sim \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$, the accuracy of the Q-function regression targets depends on the estimate of the Q-value for actions that are outside of the distribution of actions that the Q-function was ever trained on. When $\pi(\mathbf{a}|\mathbf{s})$ differs substantially from $\pi_\beta(\mathbf{a}|\mathbf{s})$, this discrepancy can result in highly erroneous target Q-values. This issue is further exacerbated by the fact that $\pi(\mathbf{a}|\mathbf{s})$ is explicitly optimized to maximize $\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[Q^\pi(\mathbf{s}, \mathbf{a})]$. This means that, if the policy can produce out-of-distribution actions for which the learned Q-function erroneously produces excessively large values, it will learn to do so. This is true whether the policy is represented explicitly, as in actor-critic algorithms, or implicitly, as the greedy policy $\pi(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} = \arg\max_{\mathbf{a}'} Q^\pi(\mathbf{s}, \mathbf{a}'))$. In standard online reinforcement learning, such issues are corrected naturally when the policy acts in the environment, attempting the transitions it (erroneously) believes to be good, and observing that in fact they are not. However, in the offline setting, the policy cannot correct such over-optimistic Q-values, and these errors accumulate over each iteration of training, resulting in arbitrarily poor final results.

This problem manifests itself in practice as an "unlearning" effect when running offline RL via dynamic programming. The experiments in Figure 2, from Kumar et al. (2019), show this unlearning effect. The horizontal axis corresponds to the number of gradient updates to the Q-function, and the vertical axis shows the actual return obtained by running the greedy policy for the learned Q-function. The learning curve resembles what we might expect in the case of overfitting: the return first improves, and then sharply falls as training progresses. However, this "overfitting" effect remains even as we increase the size of the dataset, suggesting that the phenomenon is distinct from classic statistical overfitting. As the Q-function is trained longer and longer, the target values become more and more erroneous, and the entire Q-function degrades.

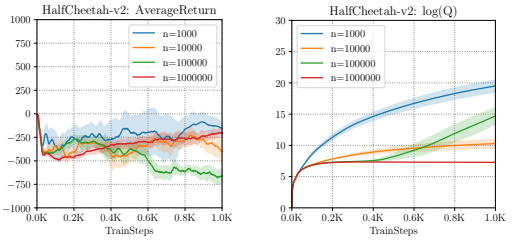

Figure 2: Performance of SAC (Haarnoja et al., 2018), an actor-critic method, on the HalfCheetah-v2 task in the offline setting, showing return as a function of gradient steps (left) and average learned Q-values on a log scale (right), for different numbers of training points (n). Note that an increase the number of samples does not generally prevent the "unlearning effect," indicating that it is distinct from overfitting. Figure from Kumar et al. (2019).

Thus, to effectively implement offline reinforcement learning via dynamic programming, it is crucial to address this out-of-distribution action problem. Previous works have observed several variants of this problem. Fujimoto et al. (2018) noted that restricting Q-value evaluation only to actions in the dataset avoids erroneous Q-values due to extrapolation error. Kumar et al. (2019) described the out-of-distribution action problem in terms of distributional shift, which suggests a less restrictive solution based on limiting distributional shift, rather than simply constraining to previously seen actions. A number of more recent works also observe that a variety of constraints can be used to mitigate action distribution shift (Wu et al., 2019a). We provide a unified view of such "policy constraint" methods in the following section.

4.3 Policy Constraints for Off-Policy Evaluation and Improvement

The basic idea behind **policy constraint** methods for offline reinforcement learning via dynamic programming is to ensure, explicitly or implicitly, that regardless of the target values in Equation (3), the distribution over actions under which we compute the target value, $\pi(\mathbf{a}'|\mathbf{s}')$, is "close" to the behavior distribution $\pi_\beta(\mathbf{a}'|\mathbf{s}')$. This ensures that the Q-function is never queried on out-of-distribution actions, which may introduce errors into the computation of the Bellman operator. If all states and actions fed into the Q-function for target value calculations are always in-distribution with respect to the Q-function training set, errors in the Q-function should not accumulate, and standard

generalization results from supervised learning should apply. Since the Q-function is evaluated on the same states as the ones on which it is trained, only the action inputs can be out of distribution, when we compute $\mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}[Q(\mathbf{s}', \mathbf{a}')]$, and therefore it is sufficient to keep $\pi(\mathbf{a}'|\mathbf{s}')$ close to $\pi_\beta(\mathbf{a}'|\mathbf{s}')$. Of course, in practice, the distributions do need to deviate in order for the learned policy to improve over the behavior policy, but by keeping this deviation small, errors due to out-of-distribution action inputs can be kept under control. The different methods in this family differ in terms of the probability metric used to define "closeness" and how this constraint is actually introduced and enforced. We can broadly categorize these methods along these two axes. We will discuss **explicit f-divergence constraints**, which directly add a constraint to the actor update to keep the policy π close to π_β in terms of an f-divergence (typically the KL-divergence), **implicit f-divergence constraints**, which utilize an actor update that, by construction, keeps π close to π_β, and **integral probability metric (IPM) constraints**, which can express constraints with more favorable theoretical and empirical properties for offline RL. Furthermore, the constraints can be enforced either as direct **policy constraints** on the actor update, or via a **policy penalty** added to the reward function or target Q-value.

Formally, we can express the family of policy iteration methods with policy constraints as a fixed point iteration involving iterative (partial) optimization of the following objectives:

$$\hat{Q}^\pi_{k+1} \leftarrow \arg\min_Q \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}} \left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}'\sim\pi_k(\mathbf{a}'|\mathbf{s}')}[\hat{Q}^\pi_k(\mathbf{s}',\mathbf{a}')] \right) \right)^2 \right]$$
$$\pi_{k+1} \leftarrow \arg\max_\pi \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[\hat{Q}^\pi_{k+1}(\mathbf{s},\mathbf{a})] \right] \text{ s.t. } D(\pi,\pi_\beta) \le \epsilon.$$

When the $\min$ and $\max$ optimizations are not performed to convergence, but instead for a limited number of gradient steps, we recover the general actor-critic method in Algorithm 3, with the exception of the constraint $D(\pi, \pi_\beta) \le \epsilon$ on the policy update. A number of prior methods instantiate this approach with different choices of D (Kumar et al., 2019; Fujimoto et al., 2018; Siegel et al., 2020). We will refer to this class of algorithms as **policy constraint** methods.

In **policy penalty** methods, the actor-critic algorithm is modified to incorporate the constraint into the Q-values, which forces the policy to not only avoid deviating from $\pi_\beta(\mathbf{a}|\mathbf{s})$ at each state, but to also avoid actions that may lead to higher deviation from $\pi_\beta(\mathbf{a}|\mathbf{s})$ at future time steps. This can be accomplished by adding a penalty term $\alpha D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s}))$ to the reward function $r(\mathbf{s},\mathbf{a})$ leading to the modified reward function $\bar{r}(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) - \alpha D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s}))$. The most well-known formulation of policy penalty methods stems from the linearly solvable MDP framework (Todorov, 2006), or equivalently the control as inference framework (Levine, 2018), where D is chosen to be the KL-divergence. An equivalent formulation adds the penalty term $\alpha D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s}))$ directly to the target Q-values and the actor objective (Wu et al., 2019a; Jaques et al., 2019), resulting in the following algorithm:

$$\hat{Q}^\pi_{k+1} \leftarrow \arg\min_Q$$
$$\mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}} \left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}'\sim\pi_k(\mathbf{a}'|\mathbf{s}')}[\hat{Q}^\pi_k(\mathbf{s}',\mathbf{a}')] - \alpha\gamma D(\pi_k(\cdot|\mathbf{s}'), \pi_\beta(\cdot|\mathbf{s}')) \right) \right)^2 \right]$$
$$\pi_{k+1} \leftarrow \arg\max_\pi \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[\hat{Q}^\pi_{k+1}(\mathbf{s},\mathbf{a})] - \alpha D(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) \right].$$

While the basic recipe for policy constraint and policy penalty methods is similar, the particular choice of how the constraints are defined and how they are enforced can make a significant difference in practice. We will discuss these choices next, as well as their tradeoffs.

Explicit f-divergence constraints. For any convex function f, the corresponding f-divergence is defined as:

$$D_f(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})} \left[f\left(\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})} \right) \right]. \tag{19}$$

KL-divergence, χ^2-divergence, and total-variation distance are some commonly used members of the f-divergence family, corresponding to different choices of function f (Nowozin et al., 2016). A variational form for the f-divergence can also be written as

$$D_f(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \max_{x:S\times A\to\mathbb{R}} \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})} \left[x(\mathbf{s},\mathbf{a}) \right] - \mathbb{E}_{\mathbf{a}'\sim\pi_\beta(\cdot|\mathbf{s})} \left[f^*(x(\mathbf{s},\mathbf{a}')) \right], \tag{20}$$

where f^* is the convex conjugate for the convex function f. Both the primal form (Equation 19) and the dual variational form (Equation 20) of the f-divergence has been used to specify policy constraints. In the dual form, an additional neural network is used to represent the function x. The standard form of "passive dynamics" in linearly solvable MDPs (Todorov, 2006) or the action prior in control as inference (Levine, 2018) corresponds to the KL-divergence (primal form), which has also been used in recent work that adapts such a KL-divergence penalty for offline reinforcement learning (Jaques et al., 2019; Wu et al., 2019a). The KL-divergence, given by $D_{\mathrm{KL}}(\pi, \pi_\beta) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[\log \pi(\mathbf{a}|\mathbf{s}) - \log \pi_\beta(\mathbf{a}|\mathbf{s})]$, can be computed by sampling action samples $\mathbf{a} \sim \pi(\cdot|\mathbf{s})$, and then computing sample-wise estimates of the likelihoods inside the expectation. It is commonly used together with "policy penalty" algorithms, by simply adding an estimate of $-\alpha \log \pi_\beta(\mathbf{a}|\mathbf{s})$ to the reward function, and employing an entropy regularized reinforcement learning algorithm, which analytically adds $\mathcal{H}(\pi(\cdot|\mathbf{s}))$ to the actor objective (Levine, 2018). One significant disadvantage of this approach is that it requires explicit estimation of the behavior policy (e.g., via behavioral cloning) to fit $\log \pi_\beta(\mathbf{a}|\mathbf{s})$.

Additionally, the sub-family of asymmetrically-relaxed f-divergences can be used for the policy constraint. For any chosen convex function f, these divergences modify the expression for D_f to integrate over only those $\mathbf{a}$ such that the density ration $\pi(\cdot|\mathbf{s})/\pi_\beta(\mathbf{a}|\mathbf{s})$ is larger than a pre-defined threshold, thereby not penalizing small density ratios. Wu et al. (2019a) briefly discuss this divergence sub-family, and we refer readers to Wu et al. (2019b) for a detailed description.

Implicit f-divergence constraints. The KL-divergence constraint can also be enforced implicitly, as in the case of AWR (Peng et al., 2019), AWAC (Nair et al., 2020), and ABM (Siegel et al., 2020). These methods first solve for the optimal next policy iterate under the KL-divergence constraint, indicated as $\bar{\pi}_{k+1}$, non-parameterically, and then project it onto the policy function class via supervised regression, implementing the following procedure:

$$\bar{\pi}_{k+1}(\mathbf{a}|\mathbf{s}) \leftarrow \frac{1}{Z}\pi_\beta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\alpha} Q_k^\pi(\mathbf{s}, \mathbf{a})\right)$$
$$\pi_{k+1} \leftarrow \arg\min_\pi D_{\mathrm{KL}}(\bar{\pi}_{k+1}, \pi)$$

In practice, the first step can be implemented by weighting samples from $\pi_\beta(\mathbf{a}|\mathbf{s})$ (i.e., the data in the buffer $\mathcal{D}$) by importance weights proportional to $\exp\left(\frac{1}{\alpha} Q_k^\pi(\mathbf{s}, \mathbf{a})\right)$, and the second step can be implemented via a weighted supervised learning procedure employing these weights. In this way, importance sampling effectively implements a KL-divergence constraint on the policy, with α corresponding to the Lagrange multiplier. The Q-function Q_k^π corresponding to the previous policy π_k can be estimated with any Q-value or advantage estimator. We refer the reader to related work for a full derivation of this procedure (Peng et al., 2019; Nair et al., 2020).

Integral probability metrics (IPMs). Another choice of the policy constraint D is an integral probability metric, which is a divergence measure with the following functional form dependent on a function class Φ:

$$D_\Phi(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \sup_{\phi\in\Phi, \phi: S\times A\to\mathbb{R}} \left|\mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[\phi(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{a}'\sim\pi_\beta(\cdot|\mathbf{s})}[\phi(\mathbf{s}, \mathbf{a}')]\right|. \tag{21}$$

Different choices for the function class Φ give rise to different divergences. For example, when Φ consists of all functions with a unit Hilbert norm in a reproducing kernel Hilbert space (RKHS), D_Φ represents the maximum mean discrepancy (MMD) distance, which can alternatively be computed directly from samples as following:

$$\mathrm{MMD}^2(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s}), \mathbf{a}'\sim\pi(\cdot|\mathbf{s})}\left[k(\mathbf{a}, \mathbf{a}')\right] -$$
$$2\mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s}), \mathbf{a}'\sim\pi_\beta(\cdot|\mathbf{s})}\left[k(\mathbf{a}, \mathbf{a}')\right] + \mathbb{E}_{\mathbf{a}\sim\pi_\beta(\cdot|\mathbf{s}), \mathbf{a}'\sim\pi_\beta(\cdot|\mathbf{s})}\left[k(\mathbf{a}, \mathbf{a}')\right],$$

where $k(\cdot, \cdot)$ represents any radial basis kernel, such as the Gaussian or Laplacian kernel. As another example, when Φ consists of all functions ϕ with a unit Lipschitz constant, then D_Φ is equivalent to the first order Wasserstein (W_1) or Earth-mover's distance:

$$W_1(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \sup_{f, ||f||_L\leq 1} \left|\mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[f(\mathbf{a})] - \mathbb{E}_{\mathbf{a}\sim\pi_\beta(\cdot|\mathbf{s}}[f(\mathbf{a})]\right|. \tag{22}$$

These metrics are appealing because they can often be estimated with non-parametric estimators. We refer the readers to Sriperumbudur et al. (2009) for a detailed discussion on IPMs. BEAR (Kumar et al., 2019) uses the MMD distance to represent the policy constraint, while Wu et al. (2019a) evaluates an instantiation of the first order Wasserstein distance.

Tradeoffs between different constraint types. The KL-divergence constraint, as well as other f-divergences, represent a particularly convenient class of constraints, since they readily lend themselves to be used in a policy penalty algorithm by simply augmenting the reward function according to $\bar{r}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) - \alpha f(\pi(\mathbf{a}|\mathbf{s})/\pi_\beta(\mathbf{a}|\mathbf{s}))$, with the important special case of the KL-divergence corresponding to $\bar{r}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \alpha \log \pi_\beta(\mathbf{a}|\mathbf{s}) - \alpha \log \pi(\mathbf{a}|\mathbf{s})$, with the last term subsumed inside the entropy regularizer $\mathcal{H}(\pi(\cdot|\mathbf{s}))$ when using a maximum entropy reinforcement learning algorithm (Levine, 2018). However, KL-divergences and other f-divergences are not necessarily ideal for offline reinforcement learning. Consider a setting where the behavior policy is uniformly random. In this case, offline reinforcement learning should, in principle, be very effective. In fact, even standard actor-critic algorithms *without* any policy constraints can perform very well in this setting, since when all actions have equal probability, there are no out-of-distribution actions (Kumar et al., 2019). However, a KL-divergence constraint in this setting would restrict the learned policy $\pi(\mathbf{a}|\mathbf{s})$ from being too concentrated, requiring the constrained algorithm to produce a highly stochastic (and therefore highly suboptimal) policy. Intuitively, an effective policy constraint should prevent the learned policy $\pi(\mathbf{a}|\mathbf{s})$ from going *outside* the set of actions that have a high probability in the data, but would not prevent it from concentrating around a *subset* of high-probability actions. This suggests that a KL-divergence constraint is not in general the ideal choice.

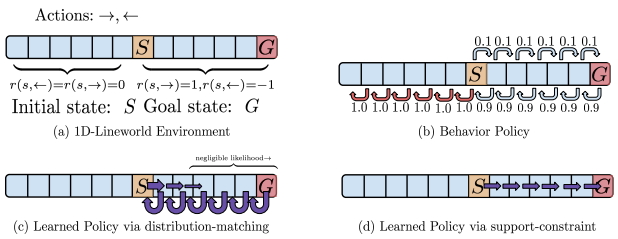

Figure 3: A comparison of support and distribution constraints on a simple 1D lineworld from Kumar (2019). The task requires moving to the goal location (marked as 'G') starting from 'S'. The behavior policy strongly prefers the left action at each state (b), such that distribution constraint is unable to find the optimal policy (c), whereas support-constraint can successfully obtain the optimal policy (d). We refer to Kumar (2019) for further discussion.

In contrast, as argued by Kumar et al. (2019) and Laroche et al. (2017), restricting the *support* of the learned policy $\pi(\mathbf{a}|\mathbf{s})$ to the support of the behavior distribution $\pi_\beta(\mathbf{a}|\mathbf{s})$ may be sufficient to theoretically and empirically obtain an effective offline RL method. In the previous example, if only the support of the learned policy is constrained to lie in the support of the behavior policy, the learned policy can be deterministic and optimal. However, when the behavior policy does not have full support, a support constraint will still prevent out-of-distribution actions. Kumar (2019) formalize this intuition and present a simple example of a 1D-lineworld environment, which we reproduce in Figure 3 where constraining distributions can lead to highly suboptimal behavior, that strongly prefers moving leftwards, thus reaching the goal location with only very low likelihood over the course of multiple intermediate steps of decision making, while support constraints still allow for learning the optimal policy, since they are agnostic to the probability density function in this setting.

Which divergence metrics can be used to constrain policy supports? In a finite sample setting, the family of f-divergences measures the difference in the *probability densities*, which makes it unsuitable for support matching. In an MDP with a discrete action-space, a simple choice for constraining the support is a metric that penalizes the total amount of probability mass on out-of-distribution actions under the π distribution, as shown below:

$$D_{\text{support},\varepsilon}(\pi(\cdot|s), \pi_\beta(\cdot|s)) = \sum_{\mathbf{a} \in A, \pi_\beta(\mathbf{a}|\mathbf{s}) \leq \varepsilon} \pi(\mathbf{a}|\mathbf{s}). \tag{23}$$

This metric has been used in the context of off-policy bandits (Sachdeva et al., 2020), but not in the context of offline reinforcement learning. When the underlying MDP has a continuous action space and exact support cannot be estimated, Kumar et al. (2019) show that a finite sample estimate of the MMD distance can be used to approximately constrain supports of the learned policy. Similar to f-divergences, the MMD distance still converges to a divergence estimate between the distribution functions of its arguments. However, Kumar et al. (2019) show experimentally (Figure 7) that, in the finite sample setting, MMD resembles a support constraining metric. The intuition is that the MMD distance does not depend on the specific densities of the behavior distribution or the policy, and can be computed via a kernel-based distance on samples from each distribution, thus making it just sufficient enough for constraining supports when finite samples are used. Some variants of the

asymmetric f-divergence that asymmetrically penalize the density ratios $\pi(\cdot|\mathbf{s})/\pi_\beta(\cdot|\mathbf{s})$ can also be used to approximately constrain supports (Wu et al., 2019b,a).

4.4 Offline Approximate Dynamic Programming with Uncertainty Estimation

As discussed in Section 4.3, policy constraints can prevent out-of-distribution action queries to the Q-function when computing the target values. Aside from directly constraining the policy, we can also mitigate the effect of out-of-distribution actions by making the Q-function resilient to such queries, via effective uncertainty estimation. The intuition behind these **uncertainty-based** methods is that, if we can estimate the *epistemic* uncertainty of the Q-function, we expect this uncertainty to be substantially larger for out-of-distribution actions, and can therefore utilize these uncertainty estimates to produce conservative target values in such cases. In this section, we briefly review such uncertainty-aware formulations of offline approximate dynamic programming methods.

Formally, such methods require learning an uncertainty set or distribution over possible Q-functions from the dataset $\mathcal{D}$, which we can denote $\mathcal{P}_\mathcal{D}(Q^\pi)$. This can utilize explicit modeling of confidence sets, such as by maintaining upper and lower confidence bounds (Jaksch et al., 2010), or directly representing samples from the distribution over Q-functions, for example via bootstrap ensembles (Osband et al., 2016), or by parameterizing the distribution over Q-values using a known (e.g., Gaussian) distribution with learned parameters (O'Donoghue et al., 2018). If calibrated uncertainty sets can be learned, then we can improve the policy using a conservative estimate of the Q-function, which corresponds to the following policy improvement objective:

$$\pi_{k+1} \leftarrow \arg\max_\pi \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \underbrace{\left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\mathbb{E}_{Q^\pi_{k+1}\sim\mathcal{P}_\mathcal{D}(Q^\pi)}[Q^\pi_{k+1}(\mathbf{s},\mathbf{a})] - \alpha\text{Unc}(\mathcal{P}_\mathcal{D}(Q^\pi))\right]\right]}_{\text{conservative estimate}}, \quad (24)$$

where Unc denotes a metric of uncertainty, such that subtracting it provides a conservative estimate of the *actual* Q-function. The choice of the uncertainty metric Unc depends on the particular choice of uncertainty estimation method, and we discuss this next.

When bootstrap ensembles[3] are used to represent the Q-function, as is common in prior work (Osband et al., 2016; Eysenbach et al., 2017; Kumar et al., 2019; Agarwal et al., 2019), typical choices of 'Unc' include variance across Q-value predictions of an ensemble of Q-functions (Kumar et al., 2019), or maximizing the Q-value with respect to the worst case or all convex combinations of the Q-value predictions of an ensemble (Agarwal et al., 2019). When a parametric distribution, such as a Gaussian, is used to represent the Q-function (O'Donoghue et al., 2018), a choice of Unc, previously used for exploration, is to use a standard deviation above the mean as an optimistic Q-value estimate for policy improvement. When translated to offline RL, an analogous estimate would be to use conservative Q-values, such as one standard deviation below the mean, for policy improvement.

4.5 Conservative Q-Learning and Pessimistic Value Functions

As an alternative to imposing constraints on the policy in an actor-critic framework, an effective approach to offline RL can also be developed by regularizing the value function or Q-function directly to avoid overestimation for out-of-distribution actions (Kumar et al., 2020b). This approach can be appealing for several reasons, such as being applicable to both actor-critic and Q-learning methods, even when a policy is not represented explicitly, and avoiding the need for explicit modeling of the behavior policy. The practical effect of such a method resembles the conservative estimate in Equation (24), but without explicit uncertainty estimation. As discussed by Kumar et al. (2020b), one simple way to ensure a conservative Q-function is to modify the objective for fitting the Q-function parameters ϕ on Line 14 of the Q-learning method in Algorithm 2 or Line 14 of the actor-critic method in Algorithm 3 to add an additional *conservative penalty* term, yielding a modified objective

$$\tilde{\mathcal{E}}(B,\phi) = \alpha\mathcal{C}(B,\phi) + \mathcal{E}(B,\phi),$$

[3] It is known in the deep learning literature that, although a true bootstrap ensemble requires resampling the dataset with replacement for every bootstrap, omitting this resampling step and simply using different random initialization for every neural network in the ensemble is sufficient to differentiate the models and provide reasonable uncertainty estimates (Osband et al., 2016). In fact, previous work has generally observed little benefit from employ proper resampling (Osband et al., 2016), which technically means that all of these methods simply employ regular (non-bootstrapped) ensembles, although the quality of their uncertainty estimates is empirically similar.

where different choices for $\mathcal{C}(B, \phi)$ lead to algorithms with different properties. A basic example is the penalty $\mathcal{C}_{\text{CQL}_0}(B, \phi) = \mathbb{E}_{\mathbf{s}\sim B, \mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q_\phi(\mathbf{s}, \mathbf{a})]$. Intuitively, this penalty minimizes the Q-values at all of the states in the buffer, for actions selected according to the distribution $\mu(\mathbf{a}|\mathbf{s})$. If $\mu(\mathbf{a}|\mathbf{s})$ is chosen *adversarially*, for example by *maximizing* the penalty $\mathcal{C}_{\text{CQL}_0}(B, \phi)$, the effect is that the conservative penalty will push down on high Q-values. Note that the standard Bellman error term $\mathcal{E}(B, \phi)$ will still enforce that the Q-values obey the Bellman backup for *in-distribution* actions. Therefore, if the penalty weight α is chosen appropriately, the conservative penalty should mostly push down on Q-values for out-of-distribution actions for which the Q-values are (potentially erroneously) high, since in-distribution actions would be "anchored" by the Bellman error $\mathcal{E}(B, \phi)$. Indeed, Kumar et al. (2020b) show that, for an appropriately chosen value of α, a Q-function trained with this conservative penalty will represent a *lower bound* on the true Q-function $Q(\mathbf{s}, \mathbf{a})$ for the current policy, both in theory and in practice. This results in a provably conservative Q-learning or actor-critic algorithm. A simple and practical choice for $\mu(\mathbf{a}, \mathbf{s})$ is to use a regularized adversarial objective, such that

$$\mu = \arg\max_{\mu} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \left[\mathbb{E}_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q_\phi(\mathbf{s}, \mathbf{a})] + \mathcal{H}(\mu(\cdot|\mathbf{s})) \right].$$

This choice inherits the lower bound guarantee above, and is simple to compute. The solution to the above optimization is given by $\mu(\mathbf{a}|\mathbf{s}) \propto \exp(Q(\mathbf{s}, \mathbf{a}))$, and we can express $\mathcal{C}_{\text{CQL}_0}(B, \phi)$ under this choice of $\mu(\mathbf{a}|\mathbf{s})$ in closed form as $\mathcal{C}_{\text{CQL}_0}(B, \phi) = E_{\mathbf{s}\sim B}[\log \sum_{\mathbf{a}} \exp(Q_\phi(\mathbf{s}, \mathbf{a}))]$. This expression has a simple intuitive interpretation: the log-sum-exp is dominated by the action with the largest Q-value, and hence this type of penalty tends to minimize whichever Q-value is largest at each state. When the action space is large or continuous, we can estimate this quantity by sampling and reweighting. For example, Kumar et al. (2020b) propose sampling from the current actor (in an actor-critic algorithm) and computing importance weights.

While the approach described above has the appealing property of providing a learned Q-function $Q_\phi(\mathbf{s}, \mathbf{a})$ that lower bounds the true Q-function, and therefore provably avoids overestimation, it tends to suffer from excessive *underestimation* – that is, it is too conservative. A simple modification, which we refer to as $\mathcal{C}_{\text{CQL}_1}(B, \phi)$, is to also add a value *maximization* term to balance out the minimization term under $\mu(\mathbf{a}|\mathbf{s})$, yielding the following expression:

$$\mathcal{C}_{\text{CQL}_1}(B, \phi) = \mathbb{E}_{\mathbf{s}\sim B, \mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q_\phi(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim B}[Q_\phi(\mathbf{s}, \mathbf{a})].$$

This conservative penalty minimizes Q-values under the adversarially chosen $\mu(\mathbf{a}|\mathbf{s})$ distribution, and *maximizes* the values for state-action tuples in the batch. Intuitively, this acts to ensure that high Q-values are only assigned to in-distribution actions. When $\mu(\mathbf{a}|\mathbf{s})$ is equal to the behavior policy, the penalty is zero on average. While this penalty does *not* produce a Q-function that is a pointwise lower bound on the true Q-function, it is a lower bound in expectation under the current policy, thereby still providing appealing conservatism guarantees, while substantially reducing underestimation in practice. As shown by Kumar et al. (2020b), this variant produces the best performance in practice.

4.6 Challenges and Open Problems

As we discussed in Section 4.1, basic approximate dynamic programming algorithms can perform very poorly in the offline setting due to distributional shift, primarily due to the distributional shift of the actions due to the discrepancy between the behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$ and the current learned policy $\pi(\mathbf{a}|\mathbf{s})$, which is used in the target value calculation for the Bellman backup. We discussed how policy constraints and explicit uncertainty estimation can in principle mitigate this problem, but both approaches have a number of shortcomings.

While such explicit uncertainty-based methods are conceptually attractive, it is often very hard to obtain calibrated uncertainty estimates to evaluate $\mathcal{P}_\mathcal{D}(\hat{Q}^\pi)$ and Unc in practice, especially with modern high-capacity function approximators, such as deep neural networks. In practice, policy constraint and conservative value function methods so far seem to outperform pure uncertainty-based methods (Fujimoto et al., 2018). This might at first appear surprising, since uncertainty estimation has been a very widely used tool in another subfield of reinforcement learning – exploration. In exploration, acting *optimistically* with respect to estimated uncertainty, or utilizing posterior sampling, has been shown to be effective in practice (Osband and Van Roy, 2017). However, in the setting of offline reinforcement learning, where we instead must act *conservatively* with respect to the uncertainty set, the demands on the fidelity of the uncertainty estimates are much higher. Exploration

algorithms only require the uncertainty set to include good behavior – in addition, potentially, to a lot of bad behavior. However, offline reinforcement learning requires the uncertainty set to directly capture the trustworthiness of the Q-function, which is a much higher bar. In practice, imperfect uncertainty sets can give rise to either overly conservative estimates, which impedes learning, or overly loose estimates, which results in exploitation of out-of-distribution actions. Of course, the relatively performance of policy constraint and uncertainty-based methods may change in the future, as the community develops better epistemic uncertainty estimation techniques or better algorithms to incorporate more suitable distribution metrics into offline RL.

Policy constraint methods do however suffer from a number of challenges. First, most of these methods fit an estimated model of the behavior policy $\pi_\beta(\cdot|\mathbf{s})$ from the dataset and constrain the learned policy $\pi(\cdot|\mathbf{s})$ against this estimated behavior policy. This means that the performance of these algorithms is limited by the accuracy of estimation of the behavior policy, which may be hard in scenarios with highly multimodal behaviors, as is the case in practice with real-world problems. For example, if a unimodal Gaussian policy is used to model a highly multi-modal action distribution, averaging across different modes while fitting this behavior policy may actually be unable to prevent the learned policy, π, from choosing out-of-distribution actions. Methods that enforce the constraint *implicitly*, using only samples and without explicit behavior policy estimation, are a promising way to alleviate this limitation (Peng et al., 2019; Nair et al., 2020).

Even when the behavior policy can be estimated exactly, a number of these methods still suffer from the undesirable effects of function approximation. When neural networks are used to represent Q-functions, undesirable effects of function approximation may prevent the Q-function from learning meaningful values even when out-of-distribution actions are controlled for in the target values. For example, when the size of the dataset is limited, approximate dynamic programming algorithms tend to overfit on the small dataset, and this error is then accumulated via Bellman backups (Fu et al., 2019; Kumar et al., 2020a). Moreover, if the dataset state-action distribution is narrow, neural network training may only provide brittle, non-generalizable solutions. Unlike online reinforcement learning, where accidental overestimation errors arising due to function approximation can be corrected via active data collection, these errors cumulatively build up and affect future iterates in an offline RL setting.

Methods that estimate a conservative or pessimistic value function (Kumar et al., 2020b) present a somewhat different set of tradeoffs. While they avoid issues associated with estimating the behavior policy and can effectively avoid overestimation, they can instead suffer from underestimation and a form of overfitting: if the dataset is small, the conservative regularizer can assign values that are too low to actions that are undersampled in the dataset. Indeed, *excessive* pessimism may be one of the bigger issues preventing better performance on current benchmarks, and an important open question is how to dynamically modulate the degree of conservatism to balance the risks of overestimation against the downside of avoiding any unfamiliar action.

A further issue that afflicts both constraint-based and conservative methods is that, while the Q-function is never evaluated on out-of-distribution states during training, the Q-function is still affected by the training set state distribution $d^{\pi_\beta}(\mathbf{s})$, and this is not accounted for in current offline learning methods. For instance, when function approximation couples the Q-value at two distinct states with very different densities under $d^{\pi_\beta}(\mathbf{s})$, dynamic programming with function approximation may give rise to incorrect solutions on the state that has a lower probability $d^{\pi_\beta}(\mathbf{s})$. A variant of this issue was noted in the standard RL setting, referred to as an absence of "corrective feedback" by Kumar et al. (2020a) (see Kumar and Gupta (2020) for a short summary), and such a problem may affect offline RL algorithms with function approximation more severely, since they have no control over the data distribution at all.

Another potential challenge for all of these offline approximate dynamic programming methods is that the degree of improvement beyond the behavior policy is restricted by error accumulation. Since the error in policy performance compounds with a factor that depends on $1/(1-\gamma)^2 \approx H^2$ (Farahmand et al., 2010; Kumar et al., 2019; Kidambi et al., 2020), a small divergence from the behavior policy at each step can give rise to a policy that diverges away from the behavior distribution and performs very poorly. Besides impacting training, this issue can also lead to severe *state* distribution shift at test time, which can lead the policy to perform very poorly. Therefore, policy constraints must be strong, so as to ensure that the overall error is small. This can restrict the amount of policy improvement that can be achieved. We might expect that directly constraining the state-action marginal distribution of

the policy, such as the methods explored in recent work (Nachum et al., 2019b) might not suffer from the error accumulation issue, however, Kidambi et al. (2020) proved a lower-bound result showing that quadratic scaling with respect to horizon is inevitable in the worst case for any offline RL method. Moreover, as previously discussed in Section 3.5, representing and enforcing constraints on the state-action marginal distributions themselves requires Bellman backups, which can themselves suffer from out-of-distribution actions. Besides the worst-case dependence on the horizon, an open question that still remains is is the development of constraints that can effectively trade off error accumulation and suboptimality of the learned policy in most "average"-case MDP instances, and can be easily enforced and optimized in practice via standard optimization techniques without requiring additional function approximators to fit the behavior policy.

5 Offline Model-Based Reinforcement Learning

The use of predictive models can be a powerful tool for enabling effective offline reinforcement learning. Since model-based reinforcement learning algorithms primarily rely on their ability to estimate $T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ via a parameterized model $T_\psi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, rather than performing dynamic programming or importance sampling, they benefit from convenient and powerful supervised learning methods when fitting the model, allowing them to effectively utilize large and diverse datasets. However, like dynamic programming methods, model-based offline RL methods are not immune to the effects of distribution shift. In this section, we briefly discuss how distributional shift affects model-based reinforcement learning methods, then survey a number of works that utilize models for offline reinforcement learning, and conclude with a brief discussion of open challenges. A complete treatment of all model-based reinforcement learning work, as well as work that learns predictive models of physics but does not utilize them for control (e.g., Lerer et al. (2016); Battaglia et al. (2016)), is outside the scope of this tutorial, and we focus primarily on work that performs both model-fitting and control from offline data.

5.1 Model Exploitation and Distribution Shift

As discussed in Section 2.1, the model in a model-based RL algorithm can be utilized either for planning (including online, via MPC) or for training a policy. In both cases, the model must provide accurate predictions for state-action tuples that are more optimal with respect to the current task. That is, the model will be queried at $\mathbf{s} \sim d^\pi(\mathbf{s})$, where π is either an explicit policy, or an implicit policy produced by running planning under the model. In general $d^\pi(\mathbf{s}) \neq d^{\pi_\beta}(\mathbf{s})$, which means that the model is itself susceptible to distribution shift. In fact, the model suffers from distribution shift both in terms of the state distribution $d^\pi(\mathbf{s})$, and the action distribution $\pi(\mathbf{a}|\mathbf{s})$.

Since the policy (or action sequence, in the case of planning) is optimized to obtain the highest possible expected reward under the current model, this optimization process can lead to the policy *exploiting* the model, intentionally producing out-of-distribution states and actions at which the model $T_\psi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ erroneously predicts successor states $\mathbf{s}_{t+1}$ that lead to higher returns than the actual successor states that would be obtained in the real MDP. This *model exploitation* problem can lead to policies that produce substantially worse performance in the real MDP than what was predicted under the model. Prior work in *online* model-based RL has sought to address this issue primarily via uncertainty estimation, analogously to the uncertainty-based methods discussed in Section 4.4, but this time modeling the epistemic uncertainty of the learned model $T_\psi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. In low-dimensional MDPs, such uncertainty estimates can be produced by means of Bayesian models such as Gaussian processes (Deisenroth and Rasmussen, 2011), while for higher-dimensional problems, Bayesian neural networks and bootstrap ensembles can be utilized (Chua et al., 2018; Janner et al., 2019). Effective uncertainty estimation is generally considered an important component of modern model-based reinforcement learning methods, for the purpose of mitigating model exploitation.

Theoretical analysis of model-based policy learning can provide bounds on the error incurred from the distributional shift due to the divergence between the learned policy $\pi(\mathbf{a}|\mathbf{s})$ and the behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$ (Sun et al., 2018b; Luo et al., 2018; Janner et al., 2019). This analysis resembles the distributional shift analysis provided in the DAgger example in Section 2.4, except that now both the policy and the transition probabilities experience distributional shift. Following Janner et al. (2019), if we assume that the total variation distance (TVD) between the learned model T_ψ and true model T is bounded by $\epsilon_m = \max_t \mathbb{E}_{d_t^\pi} D_{TV}(T_\psi(s_{t+1}|s_t, a_t) \| T(s_{t+1}|s_t, a_t))$, and the TVD between π and

π_β is likewise bounded on sampled states by ϵ_π, then the true policy value $J(\pi)$ is related to the policy estimate under the model, $J_\psi(\pi)$, according to

$$J(\pi) \geq J_\psi(\pi) - \left[\frac{2\gamma r_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} + \frac{4 r_{\max} \epsilon_\pi}{1-\gamma} \right].$$

Intuitively, the second term represents error accumulation due to the distribution shift in the policy, while the first term represents error accumulation due to the distribution shift in the model. The first term also includes a dependence on ϵ_π, because policies that diverge more from π_β will lead to states that are further outside of the data distribution, which in turn will make errors in the model more likely. Janner et al. (2019) also argue that a modified model-based RL procedure that resembles Dyna Sutton (1991), where only short-horizon rollouts from the model are generated by "branching" off of states seen in the data, can mitigate this accumulation of error. This is also intuitively natural: if applying the learned model recursively leads to compounding error, then shorter rollouts should incur substantially less error.

5.2 Brief Survey of Model-Based Offline Reinforcement Learning

A natural and straightforward way to utilize model-based reinforcement learning algorithms in the offline setting is to simply train the model from offline data, with minimal modification to the algorithm. As with Q-learning and actor-critic methods, model-based reinforcement learning algorithms can be applied to the offline setting naïvely. Furthermore, many effective model-based reinforcement learning methods *already* take steps to limit model exploitation via a variety of uncertainty estimation methods (Deisenroth and Rasmussen, 2011; Chua et al., 2018), making them reasonably effective in the offline setting. Many such methods have been known to exhibit excellent performance in conventional off-policy settings, where additional data collection is allowed, but prior data is also utilized (Sutton, 1991; Watter et al., 2015; Zhang et al., 2018; Hafner et al., 2018; Janner et al., 2019). Indeed, Yu et al. (2020) showed that MBPO (Janner et al., 2019) actually attains reasonable performance on standard offline RL settings without modification, whereas naïve dynamic programming methods (e.g., soft actor-critic (Haarnoja et al., 2018)) fail to learn meaningful policies without policy constraints. This suggests that model-based RL algorithms are likely to lead to an effective class of offline reinforcement learning methods.

Offline RL with standard model-based RL methods. A number of recent works have also demonstrated effective offline learning of predictive models and their application to control in complex and high-dimensional domains, including settings with image observations. Some of these methods have directly utilized high-capacity predictive models on high-dimensional observations, such as images, directly employing for online trajectory optimization (i.e., MPC). Action-conditional convolutional neural networks (Oh et al., 2015) have been used to provide accurate, long-term prediction of behavior in Atari games and have been combined with RL to improve sample-efficiency over model-free methods (Kaiser et al., 2019b). The *visual foresight* method involves training a video prediction model to predict future image observations for a robot, conditioned on the current image and future sequence of actions. The model is represented with a recurrent neural network, and trained on large amounts of offline data, typically collected with an uninformative randomized policy (Finn and Levine, 2017; Ebert et al., 2018). More recent work has demonstrated this approach on very large and diverse datasets, collected from multiple viewpoints, many objects, and multiple robots, suggesting a high degree of generalization, though the particular behaviors are comparatively simple, typically relocating individual objects by pushing or grasping Dasari et al. (2019). A number of prior works have also employed "hybrid" methods that combine elements of model-free and model-based algorithms, predicting future rewards or reward features conditioned on a sequence of future actions, but avoiding direct prediction of future observations. A number of such methods have been explored in the conventional online RL setting (Tamar et al., 2016; Dosovitskiy and Koltun, 2016; Oh et al., 2017), and in the offline RL setting, such techniques have been applied effectively to learning navigational policies for mobile robots from previously collected data (Kahn et al., 2018, 2020).

Off-policy evaluation with models. Model learning has also been explored considerably in the domain of off-policy evaluation (OPE), as a way to reduce the variance of importance sampled estimators of the sort discussed in Section 3.1. Here, the model is used to provide a sort of baseline for the expected return inside of an importance sampled estimator, as illustrated in Equation (6) in

Section 3.1. In these settings, the model is typically trained from offline data (Jiang and Li, 2015; Thomas and Brunskill, 2016; Farajtabar et al., 2018; Wang et al., 2017).

Distribution and safe region constraints. As with the policy constraint methods in Section 4.3, model-based RL algorithms can also utilize constraints imposed on the policy or planner with the learned model. Methods that guarantee Lyapunov stability (Berkenkamp et al., 2017) of the learned model can be used to constrain policies within "safe" regions of the state space where the chance of failure is small. Another example of such an approach is provided by deep imitative models (DIMs) (Rhinehart et al., 2018), which learn a normalizing flow model over future trajectories from offline data, conditioned on a high-dimensional observation. Like the hybrid methods described above, DIMs avoid direct prediction of observations. The learned distribution over dataset trajectories can then be used to both provide predictions for a planner, and provide a distributional constraint, preventing the planner from planning trajectories that deviate significantly from the training data, thus limiting distributional shift.

Conservative model-based RL algorithms. More recently, two concurrent methods, MoREL (Kidambi et al., 2020) and MOPO (Yu et al., 2020), have proposed offline model-based RL algorithms that aim to utilize conservative value estimates to provide analytic bounds on performance. Conceptually, these methods resemble the conservative estimation approach described in Section 4.5, but instead of regularizing a value function, they modify the MDP model learned from data to induce conservative behavior. The basic principle is to provide the policy with a penalty for visiting states under the model where the model is likely to be incorrect. If the learned policy takes actions that remain in regions where the model is accurate, then the model-based estimate of the policy's value (and its gradient) will likely be accurate also. Let $u(\mathbf{s}, \mathbf{a})$ denote an *error oracle* that provides a consistent estimate of the accuracy of the model at state-action tuple $(\mathbf{s}, \mathbf{a})$. For example, as proposed by Yu et al. (2020), this oracle might satisfy the property that $D(T_\psi(s_{t+1}|s_t, a_t) \| T(s_{t+1}|s_t, a_t)) \leq u(\mathbf{s}, \mathbf{a})$ for some divergence measure $D(\cdot, \cdot)$. Then, conservative behavior can be induced either by modifying the reward function to obtain a conservative reward of the form $\tilde{r}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) - \lambda u(\mathbf{s}, \mathbf{a})$, as in MOPO (Yu et al., 2020), or by modifying the MDP under the learned model so that the agent enters an absorbing state with a low reward value when $u(\mathbf{s}, \mathbf{a})$ is below some threshold, as in MoREL (Kidambi et al., 2020). In both cases, it is possible to show that the model-based estimate of the policy's performance under the model the modified reward function or MDP structure bounds that policy's true performance in the real MDP, providing appealing theoretical guarantees on performance. Note, however, that such approaches still require a consistent estimator for the error oracle $u(\mathbf{s}, \mathbf{a})$. Prior work has used disagreement in a bootstrap ensemble to provide this estimate, but it is not guaranteed to be consistent under sampling error, and resolving this limitation is likely to be an important direction for future work.

5.3 Challenges and Open Problems

Although model-based reinforcement learning appears to be a natural fit for the offline RL problem setting, current methods for fully offline model-based reinforcement learning generally rely on explicit uncertainty estimation for the model to detect and quantify distributional shift, for example by using a bootstrap ensemble. Although uncertainty estimation for models is in some ways more straightforward than uncertainty estimation for value functions, current uncertainty estimation methods still leave much to be desired, and it seems likely that offline performance of model-based RL methods can be improved substantially by carefully accounting for distributional shift.

Model-based RL methods also present their own set of challenges: while some MDPs are easy to model accurately, others can be exceedingly difficult. Modeling MDPs with very high-dimensional image observations and long horizons is a major open problem, and current predictive modeling methods generally struggle with long-horizon prediction. Hybrid methods that combine model-based and model-free learning, for example by utilizing short rollouts (Sutton, 1991; Janner et al., 2019) or avoiding prediction of full observations (Dosovitskiy and Koltun, 2016; Oh et al., 2017; Kahn et al., 2020) offer some promise in this area.

It is also still an open theoretical question as to whether model-based RL methods even *in theory* can improve over model-free dynamic programming algorithms. The reasoning behind this question is that, although dynamic programming methods do not *explicitly* learn a model, they essentially utilize the dataset as a "non-parametric" model. Fundamentally, both dynamic programming methods and

model-based RL methods are solving *prediction* problems, with the former predicting future returns, and the latter predicting future states. Moreover, model-free methods can be modified to predict even more general quantities (Sutton et al., 2011), such as return values with different discount factors, return values for different number of steps into the future, etc. In the linear function approximation case, it is known that model-based updates and fitted value iteration updates actually produce identical iterates (Vanseijen and Sutton, 2015; Parr et al., 2008), though it is unknown whether this relationship holds under non-linear function approximation. Therefore, exploring the theoretical bounds on the optimal performance of offline model-based RL methods under non-linear function approximation, as compared to offline dynamic programming methods, remains an open problem.

6 Applications and Evaluation

In this section, we survey and discuss evaluation methods, benchmarks, and applications for offline reinforcement learning methods. As discussed in Section 1, and as we will discuss further in Section 7, it is very likely that the full potential of offline reinforcement learning methods has yet to be fully realized, and perhaps the most exciting applications of such methods are still ahead of us. Nonetheless, a number of prior works have applied offline reinforcement learning in a range of challenging domains, from safety-critical real-world physical systems to large-scale learning from logged data for recommender systems. We first discuss how offline reinforcement learning algorithms have been evaluated in prior work, and then discuss specific application domains where such methods have already made an impact.

6.1 Evaluation and Benchmarks

While individual application domains, such as recommender systems and healthcare, discussed below, have developed particular domain-specific evaluations, the general state of benchmarking for modern offline reinforcement learning research is less well established. In the absence of well-developed evaluation protocols, one approach employed in recent work is to utilize training data collected via a standard online reinforcement learning algorithm, using either the entire replay buffer for an off-policy algorithm for training (Kumar et al., 2019; Agarwal et al., 2019; Fujimoto et al., 2018), or even data from the optimal policy. However, this evaluation setting is rather unrealistic, since the entire point of utilizing offline reinforcement learning algorithms in the real world is to obtain a policy that is better than the best behavior in the dataset, potentially in settings where running reinforcement learning online is impractical due to cost or safety concerns. A simple compromise solution is to utilize data from a "suboptimal" online reinforcement learning run, for example by stopping the online process early, saving out the buffer, and using this buffer as the dataset for offline RL (Kumar et al., 2019). However, even this formulation does not fully evaluate capabilities of offline reinforcement learning methods, and the statistics of the training data have a considerable effect on the difficult of offline RL (Fu et al., 2020), including how concentrated the data distribution is around a specific set of trajectories, and how multi-modal the data is. Broader data distributions (i.e., ones where $\pi_\beta(\mathbf{a}|\mathbf{s})$ has higher entropy) are generally easier to learn from, since there are fewer out-of-distribution actions. On the other hand, highly multi-modal behavior policies can be extremely difficult to learn from for methods that require explicit estimation of $\pi_\beta(\mathbf{a}|\mathbf{s})$, as discussed in Section 4.3 and 4.6. Our recently proposed set of offline reinforcement learning benchmarks aims to provide standardized datasets and simulations that cover such difficult cases (Fu et al., 2020).

A reasonable question we might ask in regard to datasets for offline RL is: in which situations might we actually expect offline RL to yield a policy that is significantly better than *any* trajectory in the training set? While we cannot expect offline RL to discover actions that are better than any action illustrated in the data, we can expect it to effectively utilize the compositional structure inherent in any temporal process. This idea is illustrated in Figure 4: if the dataset contains a subsequence illustrating a way to arrive at state 2 from state 1, as well as a separate subsequence illustrating how to arrive at state 3 from state 2, then an effective offline RL method should be able to learn how to arrive at state 3 from state 1, which might provide for a substantially higher

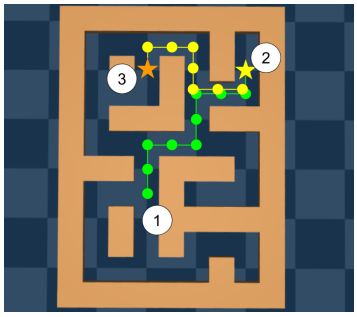

Figure 4: An example of exploiting compositional structure in trajectories to find shortest paths in the Maze2D environment in the D4RL benchmark suite (Fu et al., 2020).

final reward than any of the subsequences in the dataset. When we also consider the capacity of neural networks to generalize, we could imagine this sort of "transitive induction" taking place on a *portion* of the state variables, effectively inferring potentially optimal behavior from highly suboptimal components. This capability can be evaluated with benchmarks that explicitly provide data containing this structure, and the D4RL benchmark suite provides a range of tasks that exercise this capability (Fu et al., 2020).

Accurately evaluating the performance of offline RL algorithms can be difficult, because we are typically interested in maximizing the *online* performance of an algorithm. When simulators are available, online evaluations can be cheaply performed within the simulator order to benchmark the performance of algorithms. Off-policy evaluation (OPE) can also be used to estimate the performance of policies without explicit online interaction, but it is an active area of research as discussed in Section 3.1. Nevertheless, OPE is a popular tool in areas such as online advertising (Li et al., 2010) or healthcare (Murphy et al., 2001) where online evaluation can have significant financial or safety consequences. In certain domains, human experts can be used to assess the quality of the decision-making system. For example, Jaques et al. (2019) uses crowd-sourced human labeling to judge whether dialogue generated by an offline RL agent is fluent and amicable, and Raghu et al. (2017) evaluates using a qualitative analysis based one domain experts for sepsis treatment.

6.2 Applications in Robotics

Robotics is an appealing application domain for offline reinforcement learning, since RL has the potential to automate the acquisition of complex behavioral skills for robots – particularly with raw sensory observations, such as camera images – but conducting online data collection for each robotic control policy can be expensive and impractical. This is especially true for robots that must learn to act intelligently in complex open-world environments, since the challenge of robust visual perception alone already necessitates large training sets. The ImageNet Large-Scale Visual Recognition Challenge (Russakovsky et al., 2015) stipulates a training set of 1.5 million images for open-world object recognition, and it seems reasonable that the sample complexity for a robotic RL algorithm that must act in similar real-world settings should be at least of comparable size. For this reason, utilizing previous collected data can be of critical importance in robotics.

Several prior works have explored offline RL methods for learning robotic grasping, which is a particularly appealing task offline RL methods, since it requires the ability to generalize to a wide range of objects (Pinto and Gupta, 2016; Levine et al., 2018; Kalashnikov et al., 2018; Zeng et al., 2018). Such methods have utilized approximate dynamic programming (Kalashnikov et al., 2018) (see Figure 5), as well as more domain-specific algorithms, such as a single-step bandit formulation (Pinto and Gupta, 2016). Outside of robotic grasping, Ebert et al. (2018) propose a model-based algorithm based on prediction of future video frames for learning a variety of robotic manipulation skills from offline data, while Dasari et al. (2019) expand on this approach with a large and diverse dataset of robotic interaction. Cabi et al. (2019) propose to use reward learning from human preferences combined with offline RL to provide a user-friendly method for controlling robots for object manipulation tasks. In the domain of robotic navigation, Mo et al. (2018) propose a dataset of visual indoor scenes for reinforcement learning, collected via a camera-mounted-robot, and Mandlekar et al. (2020) proposes a dataset of human demonstrations for robotic manipulation. Kahn et al. (2020) discuss how a method based on prediction of future reward signals, blending elements of model-based and model-free learning, can learn effective navigation policies from data collected in the real world using a random exploration policy. Pure model-based methods in robotics typically involve training a model on real or simulated data, and then planning within the model to produce a policy that is executed on a real system. Approaches have included using Gaussian process models

Figure 5: Large-scale robotic grasping data collection. Kalashnikov et al. (2018) describes how a dataset of over 500,000 grasp trials collected from multiple robots was used to train a vision-based grasping policy, comparing fully offline training and online fine-tuning.

for controlling blimps (Ko et al., 2007), and using linear regression (Koppejan and Whiteson, 2009) and locally-weighted Bayesian regression (Bagnell and Schneider, 2001) for helicopter control.

6.3 Applications in Healthcare

Using offline reinforcement learning in healthcare poses several unique challenges (Gottesman et al., 2018). Safety is a major concern, and largely precludes any possibility of online exploration. Datasets can also be significantly biased towards serious outcomes (Gottesman et al., 2019), since minor cases rarely require treatment, and can lead naïve agents to erroneous conclusions, for example that any drug treatment may cause death simply because it is not prescribed to otherwise healthy individuals.

The MIMIC-III dataset (Johnson et al., 2016), which contains approximately 60K medical records from ICUs, has been influential in enabling data-driven research in healthcare treatment. Q-learning methods on this dataset has been applied to problems such as the treatment of sepsis (Raghu et al., 2017) and optimizing the use of ventilators (Prasad et al., 2017). Wang et al. (2018) apply actor-critic methods on MIMIC-III to determine drug recommendations.

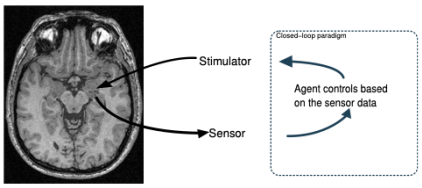

Figure 6: A real-time epilepsy treatment system, train using offline reinforcement learning (Guez et al., 2008).

Outside of ICU settings, offline RL applications include learning from recordings of seizure activity in mouse brains in order to determine optimal stimulation frequencies for reducing epileptic seizures (Guez et al., 2008). Offline RL has also been used for optimizing long term treatment plans. Shortreed et al. (2011) uses offline fitted Q-iteration for optimizing schizophrenia treatment, Nie et al. (2019) uses doubly-robust estimators to safely determine proper timings of medical treatments, and Tseng et al. (2017) uses a model-based approach for lung cancer treatment. Careful application of offline RL that can handle such challenges may offer healthcare providers powerful assistive tools for optimizing the care of patients and ultimately improving outcomes.

6.4 Applications in Autonomous Driving

As in healthcare, a significant barrier to applying reinforcement learning in the domain of self-driving vehicles is safety. In the online setting, exploratory agents can select actions that lead to catastrophic failure, potentially endangering the lives of the passengers. Thus, offline RL is potentially a promising tool for enabling, safe, effective learning in autonomous driving.

While offline RL has not yet found significant application in actual real-world self-driving vehicles (Yurtsever et al., 2020), learning-based approaches have been gaining in popularity. RobotCar (Maddern et al., 2017) and BDD-100K (Yu et al., 2018) are both large video datasets containing thousands of hours of real-life driving activity. Imitation learning has been a popular approach towards end-to-end, data-driven methods in autonomous driving (Bojarski et al., 2016; Sun et al., 2018a; Pan et al., 2017; Codevilla et al., 2018). Reinforcement learning approaches have been applied in both simulation (Sallab et al., 2017) and in the real world, with human interventions in case the vehicle violates a safety constraint (Kendall et al., 2019) (see Figure 7). Model-based reinforcement learning methods that employ constraints to keep the agent close to the training data for the model, so as to avoid out-of-distribution inputs as discussed in Section 5, can effectively provide elements of imitation learning when training on driving demonstration data, as for example in the case of deep imitative models (DIMs) (Rhinehart et al., 2018). Indeed, with the widespread availability of high-quality demonstration data, it is likely that effective methods for offline RL in the field of autonomous driving will, explicitly or implicitly, combine elements of imitation learning and reinforcement learning.

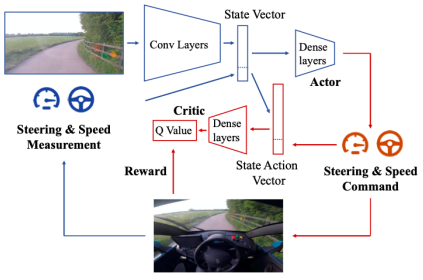

Figure 7: A road following system trained end-to-end via reinforcement learning (Kendall et al., 2019).

6.5 Applications in Advertising and Recommender Systems

Recommender systems and advertising are particularly suitable domains for offline RL because data collection is easy and efficient, and can be obtained by logging user behavior. However, these domains are also "safety critical," in the sense that a highly suboptimal policy may result in large monetary losses, thereby making unconstrained online exploration problematic. Thus, offline RL approaches have a long history of application in this area.

Off-policy evaluation is commonly used as a tool for performing A/B tests and estimating the performance of advertising and recommender systems without additional interaction with the environment. In contrast to the other applications discussed, policy evaluation for recommender systems is typically formulated within a contextual bandit problem (Langford et al., 2008; Li et al., 2010), where states may correspond to user history and actions correspond to recommendations. This approximation removes the need for *sequential* decision making, but can introduce approximation errors if actions have temporal dependence as in domains such as robotics or healthcare.

Applications of offline RL for recommender systems include slate and whole-page optimization (Swaminathan et al., 2017), applying doubly robust estimation to estimate website visits Dudík et al. (2014), and A/B testing for click optimization (Gilotte et al., 2018). Policy learning from logged, offline data has included studies on optimizing newspaper article click-through-rates (Strehl et al., 2010; Garcin et al., 2014), advertisement ranking on search pages (Bottou et al., 2013), and personalized ad recommendation for digital marketing (Theocharous et al., 2015; Thomas et al., 2017).

6.6 Applications in Language and Dialogue

Interaction via natural language is not typically thought of as a reinforcement learning problem, but in fact the formalism of sequential decision making can provide a powerful tool for natural language interaction: when dialogue is modeled as a *purposeful* interaction, the RL framework can in principle offer an effective mechanism for learning policies for outputting natural language responses to human interlocutors. The most direct way to utilize standard online RL methods for natural language interaction – by having machines engage in dialogue with real humans – can be exceedingly tedious, especially in the early stages of training, when the policy would produce mostly non-sensical dialogue. For this reason, offline RL offers a natural avenue to combine the optimal decision making formalism of RL with the kinds of large datasets of human-to-human conversations available in NLP.

In prior work, offline RL approaches have been applied in the areas of dialogue and language interfaces, where datasets consist of logged interactions, such as agent-customer transcripts (Zhou et al., 2017). An example of an application is dialogue management, which is typically concerned with accomplishing a specific goal, such as retrieving information. Examples of this have included applications of offline RL to the problem of flight booking (Henderson et al., 2008), restaurant information retrieval Pietquin et al. (2011), and restaurant recommendation (Kandasamy et al., 2017). Jaques et al. (2019) applies offline RL to the problem of dialogue generation, and focuses on producing natural responses that elicit positive feedback from human users. An example of an interaction with a trained agent is shown in Figure 8.

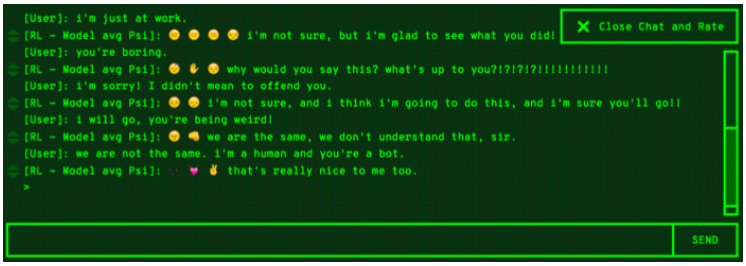

Figure 8: A dialogue agent trained via offline reinforcement learning interacting with a human user, with the aim of elicit responses with positive sentiment (Jaques et al., 2019).

7 Discussion and Perspectives

Offline reinforcement learning offers the possibility of turning reinforcement learning – which is conventionally viewed as a fundamentally active learning paradigm – into a data-driven discipline,

such that it can benefit from the same kind of "blessing of scale" that has proven so effective across a range of supervised learning application areas (LeCun et al., 2015). However, making this possible will require new innovations that bring to bear sophisticated statistical methods and combine them with the fundamentals of sequential decision making that are conventionally studied in reinforcement learning. Standard off-policy reinforcement learning algorithms have conventionally focused on dynamic programming methods that can utilize off-policy data, as discussed in Section 2.1 and Section 4, and importance sampling methods that can incorporate samples from different sampling distributions, as discussed in Section 3.2.

However, both of these classes of approaches struggle when scaled up to complex high-dimensional function approximators, such as deep neural networks, high-dimensional state or observation spaces, and temporally extended tasks. As a result, the standard off-policy training methods in these two categories have generally proven unsuitable for the kinds of complex domains typically studied in modern deep reinforcement learning. More recently, a number of improvements for offline RL methods have been proposed that take into account the statistics of distributional shift, via either policy constraints or uncertainty estimation, such as the policy constraint formulations that we discuss in Section 4.3. These formulations have the potential to mitigate the shortcomings of early methods, by explicitly account for the key challenge in offline RL: distributional shift due to differences between the learned policy and the behavior policy.

More generally, such methods shed light on the fact that offline RL is, at its core, a counter-factual inference problem: given data that resulted from a given set of decisions, infer the consequence of a *different* set of decisions. Such problems are known to be exceptionally challenging in machine learning, because they require us to step outside of the commonly used i.i.d. framework, which assumes that test-time queries involve the same distribution as the one that produced the training data (Schölkopf, 2019). It therefore stands to reason that the initial solutions to this problem, proposed in recent work, should aim to reduce distributional shift, either by constraining the policy's deviation from the data, or by estimating (epistemic) uncertainty as a measure of distributional shift. Moving forward, we might expect that a variety of tools developed for addressing distributional shift and facilitating generalization may find use in offline RL algorithms, including techniques from causal inference (Schölkopf, 2019), uncertainty estimation (Gal and Ghahramani, 2016; Kendall and Gal, 2017), density estimation and generative modeling (Kingma et al., 2014), distributional robustness (Sinha et al., 2017; Sagawa et al., 2019) and invariance (Arjovsky et al., 2019). More broadly, methods that aim to estimate and address distributional shift, constrain distributions (e.g., various forms of trust regions), and evaluate distribution support from samples are all potentially relevant to developing improved methods for offline reinforcement learning.

The counter-factual inference perspective becomes especially clear when we consider model-based offline RL algorithms, as discussed briefly in Section 5. In this case, the model answers the question: "what would the resulting state be if the agent took an action other than the one in the dataset?" Of course, the model suffers from distributional shift in much the same way as the value function, since out-of-distribution state-action tuples can result in inaccurate prediction. Nonetheless, prior work has demonstrated good results with model-based methods, particularly in regard to generalization with real-world data (Finn and Levine, 2017; Ebert et al., 2018), and a range of works on predicting physical phenomena have utilized offline datasets (Lerer et al., 2016; Battaglia et al., 2016). This suggests that model learning may be an important component of effective future offline reinforcement learning methods.

To conclude our discussion of offline reinforcement learning, we will leave the reader with the following thought. While the machine learning community frequently places considerable value on design of novel algorithms and theory, much of the amazing practical progress that we have witnessed over the past decade has arguably been driven just as much by advances in datasets as by advances in methods. Indeed, widely deployed techniques in computer vision and NLP utilize learning methods that are relatively old and well understood, and although improvements in architectures and models have driven rapid increase in performance, the increasing size and diversity of datasets – particularly in real-world applications – have been an instrumental driver of progress. In real-world applications, collecting large, diverse, representative, and well-labeled datasets is often far more important than utilizing the most advanced methods. In the standard active setting in which most reinforcement learning methods operate, collecting large and diverse datasets is often impractical, and in many applications, including safety-critical domains such as driving, and human-interactive domains such as dialogue systems, it is prohibitively costly in terms of time, money, and safety considerations.

Therefore, developing a new generation of *data-driven* reinforcement learning may usher in a new era of progress in reinforcement learning, both by making it possible to bring it to bear on a range of real-world problems that have previously been unsuited for such methods, and by enabling current applications (e.g., in robotics or autonomous driving) to benefit from much larger, more diverse, and more representative datasets that can be reused effectively across experiments.

References

Agarwal, R., Schuurmans, D., and Norouzi, M. (2019). An optimistic perspective on offline reinforcement learning. *arXiv preprint arXiv:1907.04543*.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2019). Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Bagnell, J. A. and Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510.

Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Portugaly, E., Ray, D., Simard, P., and Snelson, E. (2013). Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research*, 14(1):3207–3260.

Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., Żołna, K., Aytar, Y., Budden, D., Vecerik, M., et al. (2019). A framework for data-driven robotics. *arXiv preprint arXiv:1909.12200*.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., and Fox, D. (2019). Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE.

Chen, Y. and Wang, M. (2016). Stochastic primal-dual methods and sample complexity of reinforcement learning. *arXiv preprint arXiv:1612.02516*.

Cheng, C.-A., Yan, X., and Boots, B. (2019). Trajectory-wise control variates for variance reduction in policy gradient methods. *arXiv preprint arXiv:1908.03263*.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765.

Codevilla, F., Miiller, M., López, A., Koltun, V., and Dosovitskiy, A. (2018). End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE.

Dai, B., He, N., Pan, Y., Boots, B., and Song, L. (2016). Learning from conditional distributions via dual embeddings. *arXiv preprint arXiv:1607.04579*.

Dai, B., Shaw, A., He, N., Li, L., and Song, L. (2017a). Boosting the actor with dual critic. *arXiv preprint arXiv:1712.10282*.

Dai, B., Shaw, A., Li, L., Xiao, L., He, N., Liu, Z., Chen, J., and Song, L. (2017b). Sbeed: Convergent reinforcement learning with nonlinear function approximation. *arXiv preprint arXiv:1712.10285*.

Dasari, S., Ebert, F., Tian, S., Nair, S., Bucher, B., Schmeckpeper, K., Singh, S., Levine, S., and Finn, C. (2019). Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Dosovitskiy, A. and Koltun, V. (2016). Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Dudík, M., Erhan, D., Langford, J., Li, L., et al. (2014). Doubly robust policy evaluation and optimization. *Statistical Science*, 29(4):485–511.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. (2018). Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

Eysenbach, B., Gu, S., Ibarz, J., and Levine, S. (2017). Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*.

Farahmand, A.-m., Szepesvári, C., and Munos, R. (2010). Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, pages 568–576.

Farajtabar, M., Chow, Y., and Ghavamzadeh, M. (2018). More robust doubly robust off-policy evaluation. *arXiv preprint arXiv:1802.03493*.

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE.

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

Fu, J., Kumar, A., Soh, M., and Levine, S. (2019). Diagnosing bottlenecks in deep Q-learning algorithms. *arXiv preprint arXiv:1902.10250*.

Fujimoto, S., Meger, D., and Precup, D. (2018). Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., and Huber, A. (2014). Offline and online evaluation of news recommender systems at swissinfo. ch. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 169–176.

Gelada, C. and Bellemare, M. G. (2019). Off-policy deep reinforcement learning by bootstrapping the covariate shift. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3647–3655.

Ghavamzadeh, M., Petrik, M., and Chow, Y. (2016). Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306.

Gilotte, A., Calauzènes, C., Nedelec, T., Abraham, A., and Dollé, S. (2018). Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206.

Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., Sontag, D., Doshi-Velez, F., and Celi, L. A. (2019). Guidelines for reinforcement learning in healthcare. *Nat Med*, 25(1):16–18.

Gottesman, O., Johansson, F., Meier, J., Dent, J., Lee, D., Srinivasan, S., Zhang, L., Ding, Y., Wihl, D., Peng, X., et al. (2018). Evaluating reinforcement learning algorithms in observational health settings. *arXiv preprint arXiv:1805.12298*.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017a). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

Gu, S. S., Lillicrap, T., Turner, R. E., Ghahramani, Z., Schölkopf, B., and Levine, S. (2017b). Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in neural information processing systems*, pages 3846–3855.

Guez, A., Vincent, R. D., Avoli, M., and Pineau, J. (2008). Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *AAAI*, pages 1671–1678.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *arXiv*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*.

Hafner, R. and Riedmiller, M. (2011). Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169.

Hallak, A. and Mannor, S. (2017). Consistent on-line off-policy evaluation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1372–1383. JMLR. org.

Hallak, A., Tamar, A., and Mannor, S. (2015). Emphatic td bellman operator is a contraction. *arXiv preprint arXiv:1508.03411*.

Hallak, A., Tamar, A., Munos, R., and Mannor, S. (2016). Generalized emphatic temporal difference learning: Bias-variance analysis. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Hanna, J. P., Stone, P., and Niekum, S. (2017). Bootstrapping with models: Confidence intervals for off-policy evaluation. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952.

Henderson, J., Lemon, O., and Georgila, K. (2008). Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Computational Linguistics*, 34(4):487–511.

Huang, J. and Jiang, N. (2019). From importance sampling to doubly robust policy gradient. *arXiv preprint arXiv:1910.09066*.

Imani, E., Graves, E., and White, M. (2018). An off-policy policy gradient theorem using emphatic weightings. In *Advances in Neural Information Processing Systems*, pages 96–106.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509.

Jaques, N., Ghandeharioun, A., Shen, J. H., Ferguson, C., Lapedriza, A., Jones, N., Gu, S., and Picard, R. (2019). Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*.

Jiang, N. and Li, L. (2015). Doubly robust off-policy value evaluation for reinforcement learning. *arXiv preprint arXiv:1511.03722*.

Johnson, A. E., Pollard, T. J., Shen, L., Li-wei, H. L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. (2016). Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035.

Kahn, G., Abbeel, P., and Levine, S. (2020). Badgr: An autonomous self-supervised learning-based navigation system. *arXiv preprint arXiv:2002.05700*.

Kahn, G., Villaflor, A., Abbeel, P., and Levine, S. (2018). Composable action-conditioned predictors: Flexible off-policy learning for robot navigation. *arXiv preprint arXiv:1810.07167*.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019a). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019b). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 2.

Kakade, S. M. (2002). A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673.

Kallus, N. and Uehara, M. (2019a). Efficiently breaking the curse of horizon: Double reinforcement learning in infinite-horizon processes. *arXiv preprint arXiv:1909.05850*.

Kallus, N. and Uehara, M. (2019b). Intrinsically efficient, stable, and bounded off-policy evaluation for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3320–3329.

Kandasamy, K., Bachrach, Y., Tomioka, R., Tarlow, D., and Carter, D. (2017). Batch policy gradient methods for improving neural conversation models. *arXiv preprint arXiv:1702.03334*.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2019). Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589.

Ko, J., Klein, D. J., Fox, D., and Haehnel, D. (2007). Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Proceedings 2007 ieee international conference on robotics and automation*, pages 742–747. IEEE.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Koppejan, R. and Whiteson, S. (2009). Neuroevolutionary reinforcement learning for generalized helicopter control. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 145–152.

Kumar, A. (2019). Data-driven deep reinforcement learning. https://bair.berkeley.edu/blog/2019/12/05/bear/. BAIR Blog.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771.

Kumar, A. and Gupta, A. (2020). Does on-policy data collection fix errors in reinforcement learning? https://bair.berkeley.edu/blog/2020/03/16/discor/. BAIR Blog.

Kumar, A., Gupta, A., and Levine, S. (2020a). Discor: Corrective feedback in reinforcement learning via distribution correction. *arXiv preprint arXiv:2003.07305*.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020b). Conservative q-learning for offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149.

Lange, S., Gabel, T., and Riedmiller, M. (2012). Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer.

Langford, J., Strehl, A., and Wortman, J. (2008). Exploration scavenging. In *Proceedings of the 25th international conference on Machine learning*, pages 528–535.

Laroche, R., Trichelair, P., and Combes, R. T. d. (2017). Safe policy improvement with baseline bootstrapping. *arXiv preprint arXiv:1712.06924*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lee, D. and He, N. (2018). Stochastic primal-dual q-learning. *arXiv preprint arXiv:1810.08298*.

Lerer, A., Gross, S., and Fergus, R. (2016). Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*.

Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670.

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

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Liu, Q., Li, L., Tang, Z., and Zhou, D. (2018). Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Advances in Neural Information Processing Systems*, pages 5356–5366.

Liu, Y., Swaminathan, A., Agarwal, A., and Brunskill, E. (2019). Off-policy policy gradient with state distribution correction. *arXiv preprint arXiv:1904.08473*.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. (2018). Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*.

Maddern, W., Pascoe, G., Linegar, C., and Newman, P. (2017). 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15.

Mandlekar, A., Xu, D., Martín-Martín, R., Savarese, S., and Fei-Fei, L. (2020). Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mo, K., Li, H., Lin, Z., and Lee, J.-Y. (2018). The adobeindoornav dataset: Towards deep reinforcement learning based real-world indoor robot visual navigation. *arXiv preprint arXiv:1802.08824*.

Mousavi, A., Li, L., Liu, Q., and Zhou, D. (2020). Black-box off-policy estimation for infinite-horizon reinforcement learning. *arXiv preprint arXiv:2003.11126*.

Murphy, S. A., van der Laan, M. J., Robins, J. M., and Group, C. P. P. R. (2001). Marginal mean models for dynamic regimes. *Journal of the American Statistical Association*, 96(456):1410–1423.

Nachum, O., Chow, Y., Dai, B., and Li, L. (2019a). Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. In *Advances in Neural Information Processing Systems*, pages 2315–2325.

Nachum, O. and Dai, B. (2020). Reinforcement learning via fenchel-rockafellar duality. *arXiv preprint arXiv:2001.01866*.

Nachum, O., Dai, B., Kostrikov, I., Chow, Y., Li, L., and Schuurmans, D. (2019b). Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*.

Nadjahi, K., Laroche, R., and Combes, R. T. d. (2019). Safe policy improvement with soft baseline bootstrapping. *arXiv preprint arXiv:1907.05079*.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE.

Nair, A., Dalal, M., Gupta, A., and Levine, S. (2020). Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*.

Nie, X., Brunskill, E., and Wager, S. (2019). Learning when-to-treat policies. *arXiv preprint arXiv:1905.09751*.

Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034.

Osband, I. and Van Roy, B. (2017). Why is posterior sampling better than optimism for reinforcement learning? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2701–2710. JMLR. org.

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. (2018). The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845.

Pan, Y., Cheng, C.-A., Saigol, K., Lee, K., Yan, X., Theodorou, E., and Boots, B. (2017). Agile autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*.

Pankov, S. (2018). Reward-estimation variance elimination in sequential decision processes. *arXiv preprint arXiv:1811.06225*.

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2019). Pipps: Flexible model-based policy search robust to the curse of chaos. *arXiv preprint arXiv:1902.01240*.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 752–759.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. (2019). Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience. *arXiv preprint cs/0204043*.

Pietquin, O., Geist, M., Chandramohan, S., and Frezza-Buet, H. (2011). Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)*, 7(3):1–21.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE.

Prasad, N., Cheng, L.-F., Chivers, C., Draugelis, M., and Engelhardt, B. E. (2017). A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. *arXiv preprint arXiv:1704.06300*.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80.

Precup, D., Sutton, R. S., and Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424.

Raghu, A., Komorowski, M., Ahmed, I., Celi, L., Szolovits, P., and Ghassemi, M. (2017). Deep reinforcement learning for sepsis treatment. *arXiv preprint arXiv:1711.09602*.

Rhinehart, N., McAllister, R., and Levine, S. (2018). Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*.

Riedmiller, M. (2005). Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 661–668.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

Sachdeva, N., Su, Y., and Joachims, T. (2020). Off-policy bandits with deficient support.

Sadeghi, F. and Levine, S. (2017). CAD2RL: Real single-image flight without a single real image. In *Robotics: Science and Systems*.

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. (2019). Distributionally robust neural networks. In *International Conference on Learning Representations*.

Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76.

Schölkopf, B. (2019). Causality for machine learning. *arXiv preprint arXiv:1911.10500*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shortreed, S. M., Laber, E., Lizotte, D. J., Stroup, T. S., Pineau, J., and Murphy, S. A. (2011). Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine learning*, 84(1-2):109–136.

Siegel, N. Y., Springenberg, J. T., Berkenkamp, F., Abdolmaleki, A., Neunert, M., Lampe, T., Hafner, R., and Riedmiller, M. (2020). Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Sinha, A., Namkoong, H., and Duchi, J. (2017). Certifying some distributional robustness with principled adversarial training. *arXiv preprint arXiv:1710.10571*.

Sriperumbudur, B. K., Fukumizu, K., Gretton, A., Schölkopf, B., and Lanckriet, G. R. (2009). On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*.

Strehl, A., Langford, J., Li, L., and Kakade, S. M. (2010). Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*, pages 2217–2225.

Sun, L., Peng, C., Zhan, W., and Tomizuka, M. (2018a). A fast integrated planning and control framework for autonomous driving via imitation learning. In *Dynamic Systems and Control Conference*, volume 51913, page V003T37A012. American Society of Mechanical Engineers.

Sun, W., Gordon, G. J., Boots, B., and Bagnell, J. (2018b). Dual policy iteration. In *Advances in Neural Information Processing Systems*, pages 7059–7069.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000.

Sutton, R. S., Mahmood, A. R., and White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768.

Swaminathan, A., Krishnamurthy, A., Agarwal, A., Dudik, M., Langford, J., Jose, D., and Zitouni, I. (2017). Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*, pages 3632–3642.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. (2018). Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*.

Tang, Z., Feng, Y., Li, L., Zhou, D., and Liu, Q. (2019). Doubly robust bias reduction in infinite horizon off-policy estimation. *arXiv preprint arXiv:1910.07186*.

Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.

Theocharous, G., Thomas, P. S., and Ghavamzadeh, M. (2015). Personalized ad recommendation systems for life-time value optimization with guarantees. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Thomas, P. (2014). Bias in natural actor-critic algorithms. In *International Conference on Machine Learning (ICML)*.

Thomas, P. and Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. (2015). High-confidence off-policy evaluation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Thomas, P. S., Theocharous, G., Ghavamzadeh, M., Durugkar, I., and Brunskill, E. (2017). Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. In *Twenty-Ninth IAAI Conference*.

Todorov, E. (2006). Linearly-solvable markov decision problems. In *Advances in Neural Information Processing Systems (NIPS)*.

Tseng, H.-H., Luo, Y., Cui, S., Chien, J.-T., Ten Haken, R. K., and El Naqa, I. (2017). Deep reinforcement learning for automated radiation adaptation in lung cancer. *Medical physics*, 44(12):6690–6705.

Uehara, M. and Jiang, N. (2019). Minimax weight and q-function learning for off-policy evaluation. *arXiv preprint arXiv:1910.12809*.

Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*.

Vanseijen, H. and Sutton, R. (2015). A deeper look at planning as learning from replay. In *International conference on machine learning*, pages 2314–2322.

Wang, L., Zhang, W., He, X., and Zha, H. (2018). Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2447–2456.

Wang, M. and Chen, Y. (2016). An online primal-dual method for discounted markov decision processes. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4516–4521. IEEE.

Wang, Y.-X., Agarwal, A., and Dudik, M. (2017). Optimal and adaptive off-policy evaluation in contextual bandits. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3589–3597. JMLR. org.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754.

Wen, J., Dai, B., Li, L., and Schuurmans, D. (2020). Batch stationary distribution estimation. *arXiv preprint arXiv:2003.00722*.

Wu, Y., Tucker, G., and Nachum, O. (2019a). Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*.

Wu, Y., Winston, E., Kaushik, D., and Lipton, Z. (2019b). Domain adaptation with asymmetrically-relaxed distribution alignment. *arXiv preprint arXiv:1903.01689*.

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., and Darrell, T. (2018). Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*.

Yu, H. (2015). On convergence of emphatic temporal-difference learning. In *Conference on Learning Theory*, pages 1724–1751.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C., and Ma, T. (2020). Mopo: Model-based offline policy optimization. In *Neural Information Processing Systems (NeurIPS)*.

Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. (2020). A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*.

Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2018). Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M. J., and Levine, S. (2018). Solar: deep structured representations for model-based reinforcement learning. *arXiv preprint arXiv:1808.09105*.

Zhang, R., Dai, B., Li, L., and Schuurmans, D. (2020a). Gendice: Generalized offline estimation of stationary values. In *International Conference on Learning Representations*.

Zhang, S., Boehmer, W., and Whiteson, S. (2019). Generalized off-policy actor-critic. In *Advances in Neural Information Processing Systems*, pages 1999–2009.

Zhang, S., Liu, B., and Whiteson, S. (2020b). Gradientdice: Rethinking generalized offline estimation of stationary values. *arXiv preprint arXiv:2001.11113*.

Zhou, L., Small, K., Rokhlenko, O., and Elkan, C. (2017). End-to-end offline goal-oriented dialog policy learning via policy gradient. *arXiv preprint arXiv:1712.02838*.