Improving Sample Efficiency of Imitation and Reinforcement Learning

by

Ilya Kostrikov

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2021

Professor Rob Fergus

Acknowledgements

First, I would like to thank my advisor Rob Fergus for supporting and guiding me throughout my Ph.D. program. I would also like to thank Ofir Nachum and Jonathan Tompson, who were advising me during several research internships at Google and during the Google Student Research Advising program. I immensely enjoyed working with Rob, Ofir, and Jonathan over all these years and grateful for their encouragement and inspiration. I would like to thank George Tucker, Lerrel Pinto and Kyunghyun Cho for serving in my thesis committee.

I have met many brilliant mentors, collaborators and colleagues through my time at NYU, Facebook AI Research and Google Brain. Thanks to Elman Mansimov, Ilya Kulikov, Roberta Raileanu, Will Whitney, Jake Zhao, Kianté Brantley, Isaac Henrion, Cinjon Resnick, David Brandfonbrener, Aaron Zweig, Sainbayar Sukhbaatar, Alex Rives, Jason Lee, Krzysztof Geras, Jaan Altosaar, Tobias Weyand, James Philbin, Tom Le Paine, Dumitru Erhan, Sergey Levine, Justin Fu, Aviral Kumar, Shane Gu, Brian Cheung, Laura Graesser, Debidatta Dwibedi, Denis Zorin, Mikael Henaff, Mark Goldstein, Martin Arjovsky, and Soumith Chintala for all the insightful discussions.

I would like to thank my family for their encouragement and support. I am especially grateful to my mother for her continuous support. Thanks to my wife, Liubov for supporting me through the course of my PhD. This work would be impossible without her encouragement and patience.

Abstract

Reinforcement Learning (RL) is an area of machine learning focused on learning to make a sequence of actions in an environment that maximizes cumulative rewards. Combined with Deep Learning, Reinforcement Learning has made significant progress over the last decade across various domains. Notable successes include achieving superhuman performance on Atari games (Mnih et al., 2013), Go (Silver et al., 2016), StarCraft II (Vinyals et al., 2019), Dota 2 (Berner et al., 2019), and various continuous control tasks (Lillicrap et al., 2015).

However, RL's success stories are often limited to games and simulations where it is possible to generate a large amount of training data. This thesis describes several methods focused on improving sample efficiency to enable a wider variety of RL applications. For the first half of the thesis, we focus on Imitation Learning, where ground truth rewards are usually unknown, and expert demonstrations define optimality. First, we introduce a method for robust and sample efficient imitation learning. We adapt an imitation learning approach where an agent tries to mimic a domain expert using a GAN-like framework (Goodfellow et al., 2014) called GAIL (Ho and Ermon, 2016). We identify two primary sources of sample inefficiency associated with this approach: on-policy RL and GAN discriminator training. We show that sample inefficiency can be mitigated by performing off-policy RL training combined with off-policy training of the discriminator. We also identify and resolve some task-specific biases associated with the family of adversarial imitation learning algorithms based on GAIL. Then, we derive a principled off-policy formulation of robust imitation learning that is entirely offline and allows us to learn a policy that imitates the expert relying only on the previously collected data. For the second half of the thesis, we focus on online and offline RL where we have

access to environment rewards. We observe that off-policy RL from pixels suffers from overfitting and propose a simple solution inspired by image augmentation techniques from Computer Vision. Finally, we introduce a method for offline RL that utilizes a pre-trained behavioral policy to improve the robustness of behavior regularization widely used in the context of offline RL. In contrast to prior work on Offline RL, this method utilizes the behavior policy to regularize the critic instead of constraining the training policy. These methods aim to improve the same efficiency of reinforcement learning and enable it for a wider variety of real-world applications.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans are capable of learning new tasks using a relatively small number of interactions with a learning environment. On the other hand, Reinforcement Learning methods often require millions (Mnih et al., 2013) or even billions (Bansal et al., 2017) of interactions with the environment that makes these methods impractical for a wide variety of applications including robotics and self-driving cars. In this thesis, we aim to develop Reinforcement Learning methods that are more data-efficient and are more suitable for real-world applications.

The recent revival of Deep Learning started from the successful applications of neural networks in Computer Vision (Ciregan et al., 2012; Ciresan et al., 2011; Krizhevsky et al., 2012) and influenced other areas of machine learning, including Speech Recognition (Hinton et al., 2012a), Natural Language Processing (Bahdanau et al., 2014; Cho et al., 2014; Mikolov et al., 2013; Sutskever et al., 2014), and Reinforcement Learning. Deep Reinforcement Learning, a combination of Deep Learning and Reinforcement Learning, has recently demonstrated impressive results on various tasks. The success stories include achieving superhuman performance on classical Atari games (Mnih et al.,

2013), defeating the world champion in Go (Silver et al., 2016), achieving professional player's level in the most popular cybersport video games (Berner et al., 2019; Vinyals et al., 2019), and mastering various continuous control tasks (Lillicrap et al., 2015). However, there is a common issue associated with all of the successes of Deep Reinforcement Learning. Namely, the aforementioned methods above are rather sample inefficient, limiting their applications to games and simulations for which we can generate an almost unlimited amount of training data. This thesis aims to improve sample efficiency in Reinforcement learning through advancements in three research areas:

- learning from domain expert datasets with Imitation Learning;
- improving sample efficiency of image-based Reinforcement Learning methods by increasing the amounts of training data with image augmentation;
- utilizing suboptimal datasets with Offline Reinforcement Learning.

Imitation learning The goal of imitation learning is to learn a policy that mimics an expert in a given task. Expert demonstrations are usually provided via a pre-collected dataset. We can either directly use this dataset to train a mapping from observations to actions (Bain and Sammut, 1995) or collect additional information from the learning environment to learn a more robust policy. The latter is usually called Interactive Imitation Learning. Early approaches for Interactive Imitation Learning involve an oracle that generates optimal actions for the set of observations induced by the training policy (Ross et al., 2011). More recent methods, called Adversarial Imitation Learning (AIL) (Ho and Ermon, 2016), utilize an adversarial scheme where an additional neural network serves as an oracle, eliminating the need for human supervision. We observe that this family of methods suffers from two main limitations: first, AIL requires us to define a

discriminator in a specific form that can unintentionally encode task-specific information that might not be available before learning; second, since the methods rely on online RL to collect additional samples, they are rather sample-inefficient, requiring sampling millions of additional transitions. Both of these issues limit AIL methods for a wide variety of applications. In this thesis, we aim to address these issues. First, we introduce an approach that removes tasks-specific reward bias via correct handling of absorbing states and improves sample inefficiency by adapting off-policy learning for RL algorithm and discriminator. Then, we derive a principled offline formulation that, on the one hand, preserves the robustness properties of AIL, while on the other hand, it does not require the agent to collect additional data.

Image augmentation Sample Efficient Reinforcement Learning from images is crucial for a variety of applications of Reinforcement Learning. Sample Efficiency is usually achieved by using off-policy training with a replay buffer (Lillicrap et al., 2015; Mnih et al., 2013). However, end-to-end training of a convolutional encoder using samples from this off-policy replay buffer has been proved challenging due to highly correlated trajectories stored in the buffer and a sparse reward signal. Inspired by the techniques used to fight overfitting in computer vision for supervised (Ciregan et al., 2012; Krizhevsky et al., 2012) and self-supervised learning (Chen et al., 2020), we introduce an image augmentation approach for Reinforcement Learning. Namely, we produce several perturbations of images sampled from the replay buffer and use an averaged estimate of critic values for learning. We test our approach on standard benchmarks for continuous and discrete control, where it demonstrates state-of-the-art performance.

Offline reinforcement learning Similarly to imitation learning, in offline reinforcement learning, we are provided with a pre-collected dataset. Nevertheless, in contrast

to imitation learning, we usually assume that this dataset stores arbitrary data relaxing the requirement of expert optimality. The goal of offline reinforcement learning is to learn a policy that outperforms the policy used to collect the dataset (Levine et al., 2020). Most offline reinforcement learning methods rely on imposing behavior constraints either on the actor or on the critic (Jaques et al., 2017; Wu et al., 2019). We argue that Fisher divergence provides a natural way to regularize the critic for reinforcement learning frameworks for continuous actions and introduce an algorithm based on this idea. We observe that when the critic is defined as a sum of log-behavior policy and an offset term, a Fisher-divergence-like penalty can be implemented as a simple gradient penalty, which is widely used for adversarial learning (Gulrajani et al., 2017). We evaluate our approach on D4RL, a popular benchmark for offline reinforcement learning, where it demonstrates the state of the art performance.

1.1 List of Contributions

- **Ilya Kostrikov**, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, Jonathan Tompson.

  Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *ICLR, 2019*.

  Source code can be found at `https://github.com/google-research/google-research/tree/master/dac`.

- **Ilya Kostrikov**, Ofir Nachum, Jonathan Tompson.

  Imitation learning via off-policy distribution matching. *ICLR, 2020*.

  Source code can be found at `https://github.com/google-research/google-research/tree/master/value_dice`.

- Denis Yarats*, **Ilya Kostrikov***, Rob Fergus.

  *Equal contribution. Author ordering determined by coin flip.

  Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *ICLR, 2021*.

  Source code can be found at `https://github.com/denisyarats/drq`.

- **Ilya Kostrikov**, Jonathan Tompson, Rob Fergus, Ofir Nachum.

  Offline Reinforcement Learning with Fisher Divergence Critic Regularization. *arXiv preprint, 2021*.

  Source code can be found at `https://github.com/google-research/google-research/tree/master/fisher_brc`.

Chapter 2

Background

In this chapter, we briefly review online and offline reinforcement and imitation learning. First, we describe general reinforcement learning and connections between policy-based and value-based reinforcement learning methods. Then, we describe imitation learning methods based on adversarial distribution matching. Finally, we describe current methods and challenges in offline reinforcement learning.

2.1 Reinforcement Learning

Reinforcement Learning is an area of Machine Learning that concerns learning policies that optimize cumulative rewards based on interactions with learning environments. We usually consider this optimization problem within the context of infinite horizon Markov Decision Processes (MDPs) (Bellman, 1957). MDPs can be described as a tuple $(\mathcal{S}, \mathcal{A}, p(\cdot|s,a), p_0(\cdot), r(s,a), \gamma)$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $p(\cdot|s,a)$ is a probability distribution that defines the environment dynamics, $r(s,a)$ is a reward function, and, finally, $\gamma \in [0,1)$ is a discount factor. The goal of RL is to find a policy

$\pi(\cdot|s)$ that maximizes cumulative returns

$$\mathop{\mathbb{E}}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | a_t \sim \pi(\cdot|s_t), s_t \sim p(\cdot|s_{t-1}, a_{t-1}), s_0 \sim p_0(\cdot)].$$

Many reinforcement learning algorithms involve estimating state-action value functions that correspond to expected cumulative returns starting from the state-action pair (s, a) and following some policy $\pi(\cdot|s)$:

$$Q_\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | a_t \sim \pi(\cdot|s_t), s_t \sim p(\cdot|s_{t-1}, a_{t-1}), a_0 = a, s_0 = s], \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{2.1}$$

We can use this formulation to define a policy that is better than all others in the long run. This policy is called an *optimal policy* and is defined as

$$\pi^*(\cdot|s) = \arg\max_{\pi} \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [Q_\pi(s, a)] \text{ for all } s \in \mathcal{S}.$$

One can also define state-only values as

$$V_\pi(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [Q_\pi(s, a)].$$

In practice, this optimization problem might not be tractable, and policy learning is done iteratively via alternating between computing the state-action value function and improving the current policy. For this iterative procedure, we can estimate each new policy's values as in eq. (2.1) via Monte Carlo sampling. However, this approach can be computationally expensive and also requires us to sample trajectories from training policy that might be suboptimal. Instead of estimating the values by unrolling the policy,

we can estimate it iteratively via TD-learning:

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha(r(s,a) + \gamma Q_\pi(s',a') - Q_\pi(s,a))$$

where (s,a,s') is sampled from any transition storage, collected from the current policy, or any other policy, and α defines the learning rate. One can demonstrate that under certain condition this iterative scheme converges to the policy values (Sutton and Barto, 2018).

Riedmiller (2005) introduced a method for fitting policy values represented by neural networks based on a combination of Deep Learning and TD-learning. We can parameterize the policy and the value function as neural networks – and call them actor and critic, respectively – to apply these methods for uncountable state or action spaces. Specific parameterization of actor and critic depends on the action and state space. We can use Multi-Layer Perceptron (MLP) for low dimensional state spaces, while for image-based inputs, we encode the observations using convolutional neural networks. The actor can be represented as a Gaussian distribution with learnable parameters for continuous actions and a neural network that outputs logits for discrete. Then, the actor can be updated using policy gradient methods, while we can fit the critic using mean squared error to bootstrapped estimates of the value function:

$$L_Q(\theta) = \mathop{\mathbb{E}}_{\substack{(s,a,s')\sim\mathcal{D} \\ a'\sim\pi(\cdot|s')}} [(Q_\theta(s,a) - y)^2]$$

where

$$y = r(s,a) + \gamma Q_\theta(s',a').$$

The optimal policy for the general reinforcement learning problem is deterministic.

However, for a variety of applications, we are interested in learning a stochastic policy. Nonetheless, we can obtain a stochastic policy via Entropy Regularization (Ziebart et al., 2008). This framework requires us to add an entropy term to the bootstrapped returns

$$y = r(s, a) + \gamma(Q_\theta(s', a') + \alpha \log \pi(a'|s'))$$

then the optimal policy can be recovered as a Boltzmann distribution or Energy-Based Model

$$\pi_{EBM}(a|s) = \frac{\exp(Q_\pi(s, a)/\alpha)}{\int_{a \in \mathcal{A}} \exp(Q_\pi(s, a)/\alpha)\, da}.$$

Computing a normalization term is intractable for continuous action spaces extensively used throughout the thesis. Thus, Haarnoja et al. (2018b) proposes approximation via a simple parameterized policy π that minimizes the KL-divergence with this Boltzmann distribution

$$\pi = \arg\min_{\pi'} D_{\mathrm{KL}} \left(\pi'(\cdot|s) \middle| \frac{\exp(Q_\pi(s, a)/\alpha)}{\int_{a \in \mathcal{A}} \exp(Q_\pi(s, a)/\alpha)\, da} \right)$$

This optimization problem can be solved without computing the normalization term using the reparametrization trick (Kingma and Welling, 2013).

We will extensively use this entropy regularized RL framework throughout the thesis. However, a further discussion of entropy regularized RL is beyond the scope of this thesis.

2.2 Imitation Learning

The goal of imitation learning is to find a policy that mimics an expert's behavior in a given task. Imitation learning can be seen as a more sample-efficient approach to policy learning than Reinforcement Learning since the policy can be trained using expert demonstrations directly instead of finding an optimal policy via exploration.

The most straightforward approach to imitation learning is behavioral cloning (BC). BC involves learning a mapping from states to actions using supervised learning. Despite its simplicity, BC demonstrated impressive results in several areas of applications. Among the most exciting examples are autonomous driving projects, such as ALVINN (Pomerleau, 1989), DAVE (Muller et al., 2006), and DAVE2 (Bojarski et al., 2016), where BC was used to learn a mapping from the sensor inputs to steering angles.

Given a set of expert demonstrations $\mathcal{D}_E$, we can fit a policy parameterized as a neural network using stochastic gradient descent with the empirical negative log-likelihood (NLL) as loss function:

$$\min_{\pi} J_{\mathrm{BC}}(\pi) := -\frac{1}{N}\sum_{k=1}^{N} \log \pi(a_k|s_k), \text{ for } (s_k, a_k) \sim \mathcal{D}_E.$$

Note that the standard NLL loss minimizes the KL-divergence between the training policy and the expert:

$$\mathop{\mathbb{E}}_{s\sim\mathcal{D}_E}[D_{\mathrm{KL}}(\pi_E(\cdot|s)\|\pi(\cdot|s))] = \mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}_E}\left[\log \frac{\pi_E(a|s)}{\pi(a|s)}\right] = -\mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}_E}\left[\log \pi(a|s)\right]. \tag{2.2}$$

However, when the learned policy is executed, all policy actions affect the distribution

of future states. If this policy makes a mistake, it might generate a distribution of states different from those induces by the expert policy. Since the distribution is different from training distribution, the policy might not generalize to these states. This shift in distribution leads to a compounding of errors. Since behavioral cloning fits only state-conditional action distributions, it ignores this issue.

However, this problem can be alleviated via iterative policy learning. We can start from an initial policy $\pi_0(\cdot|s)$, which we can pre-train with BC, and generate a set of states using this policy. Then, we can annotate this set of states with optimal actions and perform another iteration. We can use an oracle (Ross et al., 2011) to obtain this annotation. However, querying an expert might be expensive and time-consuming, prohibiting real-world applications.

Distribution matching is another way to formulate the problem of robust imitation learning. Instead of minimizing the KL-divergence between the policies as in eq. (2.2), we can consider an alternative divergence (Syed et al., 2008):

$$D_{\mathrm{KL}}\left(d^{\pi}||d^{\exp}\right) = -\mathop{\mathbb{E}}_{(s,a)\sim d^{\pi}}\left[\log \frac{d^{\pi}(s,a)}{d^{\exp}(s,a)}\right]$$

where d^{π} is a time-discounted state action distribution of the policy π. The distribution d^{π} is called occupancy metric of the policy π defined as

$$d^{\pi}(s,a) = (1-\gamma)\cdot\sum_{t=0}^{\infty}\gamma^{t}p(s_t = s, a_t = a|s_0 \sim p_0(\cdot), s_t \sim p(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)).$$

Due to the inclusion of the temporal MDP structure, this formulation is more robust than behavioral cloning; in addition to matching the KL-divergence between policies, it also

ensures that the training policy does not drift from the expert state distribution:

$$D_{\mathrm{KL}}\left(d^{\exp}(s,a)||d^{\pi}(s,a)\right) = D_{\mathrm{KL}}\left(d^{\exp}(s)||d^{\pi}(s)\right) + \underbrace{\mathop{\mathbb{E}}_{s\sim d^{\exp}}\left[D_{\mathrm{KL}}(\pi_{\exp}(a|s)||\pi(a|s))\right]}_{\text{Behavioral Cloning}}.$$

Moreover, we can also bound the absolute difference of returns of two policies:

$$|\rho(\pi_{exp}) - \rho(\pi)| \leq \sum_{s\in\mathcal{S}, a\in\mathcal{A}} |r(s,a)| \cdot |d^{exp}(s,a) - d^{\pi}(s,a)|$$

where we use the fact that returns can be alternatively expressed as an expectation of rewards with respect to the occupancy measure:

$$\rho(\pi) = \mathop{\mathbb{E}}_{s\sim p_0}\left[V_{\pi}(s)\right] = \sum_{a\in\mathcal{A}, s\in\mathcal{S}} [r(s,a)d^{\pi}(s,a)].$$

Due to these properties, occupancy matching became a dominant approach to imitation learning over the past years. Ho and Ermon (2016) proposed Generative Adversarial Imitation Learning (GAIL), a GAN-like (Goodfellow et al., 2014) approach to occupancy matching. The approach involves training an adversarial discriminator to estimate log-ratios between the expert and training policy

$$D^*(s,a) = \log \frac{d^{exp}(s,a)}{d^{\pi}(s,a)}$$

then this discriminator can be used to provide RL rewards to any off-the-shelf RL algorithm, e.g., Trust Region Policy Optimization (Schulman et al., 2015), which was used in the original paper. Similarly to GANs, GAIL does not train the discriminator to optimality but alternates between actor, critic and discriminator updates. The GAIL

framework was later extended to learn state-only rewards for imitation learning in Fu et al. (2017) and more general divergences in Ghasemipour et al. (2020).

2.3 Offline Reinforcement Learning

Offline Reinforcement Learning aims to find an optimal policy using a pre-collected dataset. In contrast to Imitation Learning, we assume that this dataset also collects reward annotations, and we are not allowed to perform additional interactions with the learning environment.

The naive approach to Offline Reinforcement Learning is to apply methods for off-policy RL since these methods use samples from a replay buffer to learn a policy. However, these methods require the dataset to have complete action coverage. In online learning, we can achieve that via exploration. On the other hand, when these methods are used for offline RL, values for unseen actions are incorrectly estimated, which results in compounding errors due to target value bootstrapping. In particular, target state-action values for Q-learning are computed as

$$y_i = r_i + \gamma \arg\max_{a \in \mathcal{A}} Q_\theta(s'_i, a). \tag{2.3}$$

Then, we can update parameters for $Q_\theta(s, a)$ using the following gradient update rule:

$$\theta \leftarrow \theta - \omega \nabla_\theta [\sum_{i=1}^{N} (Q_\theta(s_i, a_i) - y_i)^2] \text{ for } (s_i, a_i, r_i, s'_i) \sim \mathcal{D}.$$

The maximum in eq. (2.3) is taken over all actions, including those not provided in the dataset for the state s'_i. This leads to incorrect estimates of targets based on extrapolated state-action values — moreover, these errors compound due to the update rule's recursive

nature. To avoid this issue, Fujimoto et al. (2018a) proposed to restrict the maximization in eq. (2.3) only to the dataset actions:

$$y_i = r_i + \gamma \underset{\substack{a \in \mathcal{A} \\ \text{s.t. } (s'_i, a) \in \mathcal{D}}}{\arg\max} Q_\theta(s'_i, a) \text{ for } (s_i, a_i, r_i, s'_i) \sim \mathcal{D}. \tag{2.4}$$

Fujimoto et al. (2018a) implement this concept via fitting a density model to the offline dataset and taking a maximum over several samples from this density model. However, this approach might require us to sample a large number of candidate actions that makes this approach computationally inefficient.

This constraint can also be implemented in a soft fashion (Jaques et al., 2017; Kumar et al., 2019; Wu et al., 2019) by augmenting the policy learning objective with a KL-divergence term:

$$L_\pi(\theta) = \mathop{\mathbb{E}}_{s \sim \mathcal{D}} \left[\mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} Q_\theta(s, a) + \alpha D_{\mathrm{KL}}(\pi_\theta(\cdot|s) \| \pi_{\mathcal{D}}(\cdot|s)) \right].$$

Nonetheless, the methods based on constraining the policy suffer from a common problem. The policy might query values for out-of-distribution actions for which the critic is not defined for the tabular case, or fails to generalize in the function approximation case. Moreover, these undefined or extrapolated values might dominate over the imposed policy constraints.

AlgaeDICE (Nachum et al., 2019c) introduces an approach that implicitly regularizes the critic with f-Divergence between the training and behavior policy's occupancy metrics. In particular, AlgaeDICE augments the residual learning objective (Baird, 1995) for critic

with a linear term. Specifically, for $f(x) = \frac{1}{2}x^2$ AlgaeDICE objective becomes:

$$L_{ADICE}(\theta) = \mathop{\mathbb{E}}_{(s,a,r)\sim\mathcal{D}}[(r(s,a) + \gamma \mathop{\mathbb{E}}_{\substack{s'\sim p(\cdot|s,a)\\ a'\sim\pi(\cdot|s')}}[Q_\theta(s',a')] - Q_\theta(s,a))^2] + (1-\gamma) \mathop{\mathbb{E}}_{\substack{s_0\sim p_0(\cdot)\\ a_0\sim\pi(\cdot|s_0)}}[Q_\theta(s_0,a_0)].$$

The optimal critic for this objective is penalized by $-\frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)}$, where d^π and $d^\mathcal{D}$ are occupancy measures induced by the training policy π and the dataset behavioral policy respectively.

However, since AlgaeDICE relies on residual learning which does not demonstrate solid empirical results in practice despite strong theoretical properties. Conservative Q-Learning (CQL) is an alternative approach to constraining the critic and actor updates proposed by Kumar et al. (2020). Instead of using residual learning, CQL relies on TD learning, which has demonstrated strong empirical performance on a variety of tasks. Similarly to AlgaeDICE, CQL introduces a critic fitting objective that pushed down values for out-of-distribution actions while maximizing values for actions seen in the dataset:

$$L_{CQL}(\theta) = \mathop{\mathbb{E}}_{\substack{(s,a,s')\sim\mathcal{D}\\ a'\sim\pi(\cdot|s')}}[(y - Q_\theta(s,a))^2] + \alpha(\log \sum_{a\in\mathcal{A}}[\exp(Q(s,a))] - Q(s,a)).$$

Due to the change of the underlying value fitting algorithm, CQL augments the critic only with some divergence between two policies omitting the state distributions. One of the advantages of this approach is that the learned values are ready for policy learning without any additional modifications. Alternatively, the CQL objective can be seen as a

sum of the standard mean squared error TD-loss and Energy Based Model loss. We will explore this property in this thesis.

Chapter 3

Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning

We identify two issues with the family of algorithms based on the Adversarial Imitation Learning framework. The first problem is implicit bias present in the reward functions used in these algorithms. While these biases might work well for some environments, they can also lead to sub-optimal behavior in others. Secondly, even though these algorithms can learn from few expert demonstrations, they require a prohibitively large number of interactions with the environment to imitate the expert for many real-world applications. To address these issues, we propose a new algorithm called Discriminator-Actor-Critic that uses off-policy Reinforcement Learning to reduce policy-environment interaction sample complexity by an average factor of 10. Furthermore, since our reward function is designed to be unbiased, we can apply our algorithm to many problems without making any task-specific adjustments.

3.1 Introduction

The Adversarial Imitation Learning (AIL) class of algorithms learns a policy that robustly imitates an expert's actions via a collection of expert demonstrations, an adversarial discriminator, and a reinforcement learning method. For example, the Generative Adversarial Imitation Learning (GAIL) algorithm (Ho and Ermon, 2016) uses a discriminator reward and a policy gradient algorithm to imitate an expert RL policy on standard benchmark tasks. Similarly, the Adversarial Inverse Reinforcement Learning (AIRL) algorithm (Fu et al., 2017) makes use of a modified GAIL discriminator to recover a reward function that can be used to perform Inverse Reinforcement Learning (IRL) (Abbeel and Ng, 2004) and whose subsequent dense reward is robust to changes in dynamics or environment properties. Importantly, AIL algorithms such as GAIL and AIRL obtain higher performance than supervised Behavioral Cloning (BC) when using a small number of expert demonstrations, experimentally suggesting that AIL algorithms alleviate some of the distributional drift (Ross et al., 2011) issues associated with BC. However, both these AIL methods suffer from two critical issues that this work will address: 1) a large number of policy interactions with the learning environment is required for policy convergence and 2) bias in the reward function formulation and improper handling of the environment terminal states introduces implicit rewards priors that can either improve or degrade policy performance.

While GAIL requires as little as 200 expert frame transitions (from 4 expert trajectories) to learn a robust reward function on most MuJoCo (Todorov et al., 2012) tasks, the number of policy frame transitions sampled from the environment can be as high as 25 million in order to reach convergence. If PPO (Schulman et al., 2017) is used in place of TRPO (Schulman et al., 2015), the sample complexity can be reduced somewhat

(for example, as in fig. 3.4, 25 million steps reduces to approximately 10 million steps). However, it is still intractable for many robotics or real-world applications. In this work, we address this issue by incorporating an off-policy RL algorithm (TD3, Fujimoto et al. (2018b)) and an off-policy discriminator to decrease the sample complexity by many orders of magnitude.

This work will also illustrate how the specific form of AIL reward function used has a significant impact on agent performance for episodic environments. For instance, as we will show, a strictly positive reward function prevents the agent from solving tasks in a minimal number of steps, and a strictly negative reward function cannot emulate a survival bonus. Therefore, one must know the true environment reward and incorporate such priors to choose a suitable reward function for the successful application of GAIL and AIRL. We will discuss these issues in formal detail and present a simple - yet effective - solution that drastically improves policy performance for episodic environments; we explicitly handle absorbing state transitions by learning the reward associated with these states.

We propose a new algorithm called Discriminator-Actor-Critic (DAC), which is compatible with both the popular GAIL and AIRL frameworks, incorporates an explicit terminal state handling off-policy discriminator an off-policy actor-critic reinforcement learning algorithm. DAC achieves state-of-the-art AIL performance for several complex imitation learning tasks. More specifically, in this work we:

- Identify and propose solutions for the problem of bias in discriminator-based reward estimation in imitation learning.
- Accelerate learning from demonstrations by providing an off-policy variant for AIL algorithms, significantly reducing the number of agent-environment interactions.

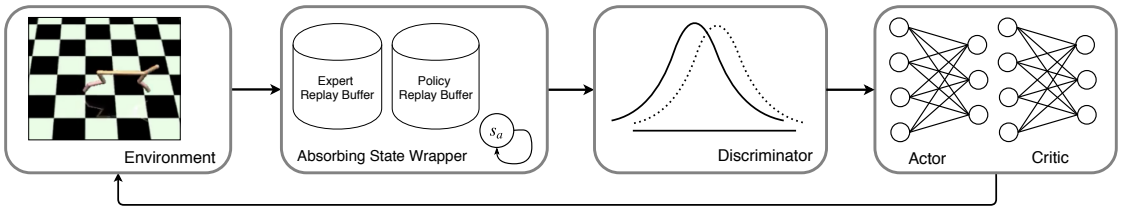


Figure 3.1: The Discriminator-Actor-Critic imitation learning framework. We first wrap the learning environment to handle absorbing state transitions correctly. Then, we train a discriminator sampling from the replay buffer. And, finally, we use the rewards produced by the discriminator within an off-policy RL algorithm.

- Illustrate the robustness of DAC to noisy, multi-modal, and constrained expert demonstrations by performing experiments with human demonstrations on non-trivial robotic tasks.

3.2 Related Work

Imitation learning has been broadly studied under the twin umbrellas of Behavioral Cloning (BC) (Bain and Sammut, 1995; Ross et al., 2011) and Inverse Reinforcement Learning (IRL) (Ng et al., 2000). To recover the underlying policy, IRL performs an intermediate step of estimating the reward function followed by RL on this function (Abbeel and Ng, 2004; Ratliff et al., 2006). Operating in the Maximum Entropy IRL formulation (Ziebart et al., 2008), Finn et al. (2016b) introduce an iterative-sampling based estimator for the partition function, deriving an algorithm for recovering non-linear reward functions in high-dimensional state and action spaces. Finn et al. (2016a), and Fu et al. (2017) further extend this by exploring the theory and practical considerations of an adversarial IRL framework and draw connections between IRL and cost learning in GANs (Goodfellow et al., 2014).

In practical scenarios, we are often interested in recovering the expert's policy rather

than the reward function. Following Syed et al. (2008), and by treating imitation learning as an occupancy matching problem, Ho and Ermon (2016) proposed a Generative Adversarial Imitation Learning (GAIL) framework for learning a policy from demonstrations, which bypasses the need to recover the expert's reward function. More recent work extends the framework by improving stability and robustness (Kim and Park, 2018; Wang et al., 2017) and making connections to model-based imitation learning (Baram et al., 2017). These approaches generally use on-policy algorithms for policy optimization, trading off sample efficiency for training stability.

Learning complex behaviors from sparse reward signals poses a significant challenge in reinforcement learning. In this context, expert demonstrations or template trajectories have been successfully used (Peters and Schaal, 2008) for initializing RL policies. There has been a growing interest in combining extrinsic sparse reward signals with imitation learning for guided exploration (Kang et al., 2018; Le et al., 2018; Vecerík et al., 2017; Zhu et al., 2018). Off-policy learning from demonstration has been previously studied under the umbrella of accelerating reinforcement learning by structured exploration (Hester et al., 2017; Nair et al., 2017) An implicit assumption of these approaches is access to demonstrations and reward from the environment; our approach requires access only to expert demonstrations.

Our work is most related to AIL algorithms (Fu et al., 2017; Ho and Ermon, 2016; Torabi et al., 2018). In contrast to Ho and Ermon (2016) which assumes *(state-action-next state)* transition tuples, Torabi et al. (2018) has weaker assumptions by relying only on observations and removing the dependency on actions. The contributions in this work are complementary (and compatible) to Torabi et al. (2018).

3.3 Background

3.3.1 Markov Decision Process

We consider problems that satisfy the definition of Markov Decision Processes (MDPs), formalized by the tuple: $(\mathcal{S}, \mathcal{A}, p(\cdot|s,a), p_0(\cdot), r(s,a), \gamma)$. Here $\mathcal{S}$, $\mathcal{A}$ represent the state and action spaces respectively, $p_0(s)$ is the initial state distribution, $p(\cdot|s,a)$ defines environment dynamics represented as a conditional state distribution, $r(s,a)$ is a reward function and γ is discount factor.

In continuing tasks, where environment interactions are unbounded in sequence length, the returns for a single trajectory $\tau = \{(s_t, a_t)\}_{t=0}^{\infty}$ are defined as

$$R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k).$$

To use the same notation for episodic tasks, whose finite length episodes end when reaching a terminal state, we can define a set of *absorbing states* s_a (Sutton et al., 1998) that an agent enters after the end of an episode. These states have zero rewards and transition to themselves for all agent actions (see fig. 3.2): $s_a \sim p(\cdot|s_T, a_T)$, $r(s_a, \cdot) = 0$ and $s_a \sim p(\cdot|s_a, \cdot)$. With this absorbing state notation, episodic task returns can be defined simply as

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k).$$

In many imitation learning and IRL algorithms, a common assumption is to assign zero reward value, often implicitly, to absorbing states. As we will discuss in detail in section 3.4.3, our DAC algorithm will assign a learned, potentially non-zero reward for absorbing states, and we will demonstrate empirically in section 3.4.2 that it is crucial to properly handle the absorbing states for algorithms that involve reward learning.

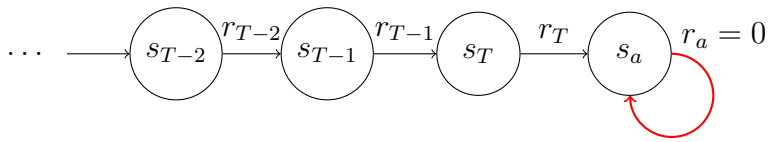


Figure 3.2: Absorbing states for episodic tasks. After reaching the terminal state s_T, the agent enters an absorbing state s_a, and loops in the absorbing state, receiving constant rewards, which are usually set to zero.

3.3.2 Adversarial Imitation Learning

To learn a robust reward function, we use the GAIL framework (Ho and Ermon, 2016). Inspired by maximum entropy IRL (Ziebart et al., 2008) and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), GAIL trains a binary classifier, $D(s, a)$, referred to as the *discriminator*, to distinguish between transitions sampled from an expert and those generated by the trained policy. In standard GAN frameworks, a generator gradient is calculated by backprop through the learned discriminator. However, in GAIL, the discriminator is used to provide rewards for the policy learning via some on-policy RL optimization scheme (e.g., TRPO (Schulman et al., 2015)):

$$\max_{\pi} \max_{D} \mathbb{E}_{\pi}[\log(D(s, a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s, a))] - \lambda H(\pi) \tag{3.1}$$

where $H(\pi)$ is an entropy regularization term.

The rewards learned by GAIL might not correspond to true rewards (Fu et al., 2017) but can be used to match the expert occupancy measure. Ho and Ermon (2016) draw analogies between distribution matching using GANs and occupancy matching with GAIL. They demonstrate that by maximizing the adversarial discriminator's rewards, the training policy matches the expert's occupancy measure with some regulation term defined by GAN's loss function choice.

In principle, GAIL can be incorporated with any on-policy RL algorithm. However,

we adapt it for off-policy training (discussed in section 3.4.4). As can be seen from eq. (3.1), the algorithm requires state-action pairs to be sampled from the learned policy. In section 3.4.4 we will discuss the algorithm's modifications for off-policy training.

3.4 Discriminator-Actor-Critic

In this section, we present the Discriminator-Actor-Critic (DAC) algorithm. This algorithm comprises two parts: a method for unbiasing adversarial reward functions, discussed in section 3.4.3, and an off-policy discriminator formulation of AIL, discussed in section 3.4.4. A high-level pictorial representation of this algorithm is also shown in fig. 3.1.

3.4.1 Bias in Reward Functions

In the following section, we present examples of bias present in reward functions in different AIL algorithms:

- In the GAIL framework, and follow-up methods, such as GMMIL (Kim and Park, 2018) and AIRL, zero rewards are implicitly assigned for absorbing states, while some learned reward function $r(s, a)$ assigns rewards for intermediate states depending on properties of a task.

- For certain environments, a survival bonus in the form of a per-step positive reward is added to the rewards received by the agent that encourages agents to survive longer in the environment. We observe that a commonly used form of the reward function, $r(s, a) = -\log(1 - D(s, a))$, works well for environments that require a survival bonus. Still, since the recovered reward function can never be negative,

it cannot recover the true reward function for environments where an agent must solve the task in a minimal number of steps. Thus, using this form of the reward function will lead to sub-optimal solutions. These positive rewards incentivize the agent to move in loops or take small actions (in continuous action spaces) that keep it close to the states in the expert's trajectories. The agent keeps collecting positive rewards without actually attempting to solve the task demonstrated by the expert (see section 3.4.1).[1]

- Another commonly used reward formulation is $r(s, a) = \log(D(s, a))$. This formulation is often used for tasks with a per-step penalty when a reward function consists of a negative constant assigned unconditionally of states and actions. However, this variant assigns only negative rewards and cannot learn a survival bonus. We demonstrate that such strong priors lead to good results even with no expert trajectories on widely used benchmarks (see fig. 3.5).

Crafting different task-dependent reward functions is undesirable for practical applications. In the next section, we describe an illustrative example of a typical failure of biased reward functions. We also propose a method to unbias the reward function in our imitation learning algorithm. It can recover different reward functions without adjusting the form of the reward function.

3.4.2 An Illustrative Example of Reward Bias

Firstly, we illustrate how $r(s, a) = -\log(1 - D(s, a))$ cannot match the expert trajectories for environments with per-step penalties. Consider a simple MDP with 3 states: s_1, s_2, s_g, where s_g is a goal state and agents receive a reward by reaching the

[1]Note that this behavior was described in the early reward shaping literature (Ng et al., 1999).

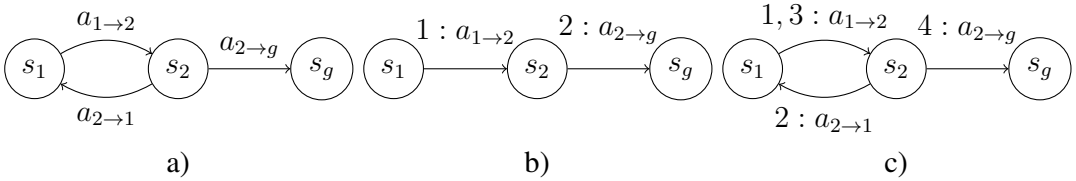


Figure 3.3: a) An MDP with 3 possible states and 3 possible actions. b) Expert demonstration. c) A policy (potentially) more optimal than the expert policy according to the GAIL reward function.

goal state, and three transitions that correspond to actions $a_{1\to2}$, $a_{2\to1}$ and $a_{2\to g}$; where $a_{i\to j}$ is such that $s_j \sim p(\cdot|s_i, a_{i\to j})$, as shown in fig. 3.3 a). And for every state the expert demonstrations are defined as $\pi_E(s_1) = a_{1\to2}$, $\pi_E(s_2) = a_{2\to g}$, (as shown in fig. 3.3 b). This expert clearly reaches the goal state in the optimal number of steps.

Now consider the trajectory of fig. 3.3 c): $(s_1, a_{1\to2}) \to (s_2, a_{2\to1}) \to (s_1, a_{1\to2}) \to (s_2, a_{2\to g})$. This trajectory has the return

$$R_\pi = r(s_1, a_{1\to2}) + \gamma r(s_2, a_{2\to1}) + \gamma^2 r(s_1, a_{1\to2}) + \gamma^3 r(s_2, a_{2\to g}).$$

While the expert return is

$$R_E = r(s_1, a_{1\to2}) + \gamma r(s_2, a_{2\to g}).$$

Assuming that we have a discriminator trained to convergence, it will assign a close to zero value to the reward $r(s_2, a_{2\to1})$ since it never appears in expert demonstrations. Therefore, from $R_\pi < R_E$ one can derive

$$r(s_1, a_{1\to2}) < \frac{(1-\gamma^2)}{\gamma} r(s_2, a_{2\to g}).$$

Thus, for the loopy trajectory to have a smaller return than our expert policy, we need

$$r(s_1, a_{1\to 2}) < \frac{0.0199}{0.99} \cdot r(s_2, a_{2\to g}),$$

for a commonly used value of the discount, $\gamma = 0.99$, in model-free RL algorithms. However, the optimal values for the GAN discriminator in this case are

$$r(s_1, a_{1\to 2}) = -log(1 - 0.5) \approx 0.3$$

and

$$r(s_2, a_{2\to g}) = -log(1 - 2/3) \approx 0.477.$$

Hence, the inequality above does not hold. As such, the convergence of GAIL to the expert policy with this reward function is possible under only certain values of γ, and this value depends heavily on the task MDP. At the same time, since the reward function is strictly positive it implicitly provides a survival bonus. In other words, regardless of how the discriminator actually classifies state-action tuples, it always rewards the policy for avoiding absorbing states (see section 3.5.2). Fundamentally, this characteristic makes it difficult to attribute policy performance to the robustness of the GAIL learned reward since the RL optimizer can often solve the task as long as the reward is strictly positive.

Another common reward variant, $r(s, a) = \log(D(s, a))$, which corresponds to the original saturating loss for GANs, penalizes every step and leads to collapsing in environments with a survival bonus. This phenomenon can be demonstrated using reasoning similar to the one stated above.

Finally, AIRL uses the reward function: $r(s, a, s') = \log(D(s, a, s') - \log(1 - D(s, a, s'))$, which can assign both positive and negative rewards for each time step. In

AIRL, as in the original GAIL, the agent receives zero rewards at the end of the episode, leading to sub-optimal policies (and training instability) in environments with a survival bonus. At the beginning of training, this reward function assigns rewards with a negative bias because it is easy for the discriminator to distinguish samples for an untrained policy and an expert. So it is common for learned agents to finish an episode earlier (to avoid additional negative penalty) instead of trying to imitate the expert.

3.4.3 Unbiasing Reward Functions

To resolve the issues described in section 3.4.2, we suggest explicitly learning rewards for absorbing states for expert demonstrations and trajectories produced by a policy. Thus, the returns for final states are defined as

$$R_T = r(s_T, a_T) + \sum_{t=T+1}^{\infty} \gamma^{t-T} r(s_a, \cdot)$$

with a learned reward $r(s_a, \cdot)$ instead of just

$$R_T = r(s_T, a_T).$$

We implemented these absorbing states by adding an extra indicator dimension that indicates whether each environment state is absorbing or not. For absorbing states, we set the indicator dimension to one and all other dimensions to zero. The GAIL discriminator can distinguish whether reaching an absorbing state is desirable behavior from the expert's perspective and assign the rewards accordingly.

Instead of recursively computing the Q values, this issue can be addressed by analyti-

cally deriving the following returns for the terminal states:

$$R_T = r(s_T, a_T) + \frac{\gamma r(s_a, \cdot)}{1 - \gamma}.$$

However, in practice, we found this alternative to be significantly less stable. Thus, we add absorbing states and corresponding transitions to the replay buffer and treat them similarly to other states.

3.4.4 Addressing Sample Inefficiency

As previously mentioned, GAIL requires a significant number of interactions with a learning environment to imitate an expert policy. To address the sample inefficiency of GAIL, we use an off-policy RL algorithm and perform off-policy training of the GAIL discriminator performed in the following way. Instead of sampling trajectories from a policy directly, we sample transitions from a replay buffer $\mathcal{R}$ collected while performing off-policy training:

$$\max_D \mathbb{E}_{\mathcal{R}}[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi). \tag{3.2}$$

The optimal discriminator defined by Equation (3.2) attempts to match the occupancy measures between the expert and the distribution induced by the replay buffer $\mathcal{R}$, which can be seen as a mixture of all policy distributions that appeared during training, instead of the latest trained policy π. In order to recover the original on-policy expectation, one needs to use importance weighting:

$$\max_D \mathbb{E}_{\mathcal{R}}\left[\frac{p_{\pi_\theta}(s,a)}{p_{\mathcal{R}}(s,a)} \log(D(s,a))\right] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi). \tag{3.3}$$

However, it can be challenging to estimate these densities correctly, and the discriminator updates might have a large variance. We found that the algorithm works well in practice with the importance weight omitted.

We use the GAIL discriminator to define rewards for training a policy using TD3; we update per-step rewards every time we pull transitions from the replay buffer using the latest discriminator. The TD3 algorithm provides a good balance between sample complexity and simplicity of implementation and so is a good candidate for practical applications. Additionally, depending on the distribution of expert demonstrations and properties of the task, off-policy RL algorithms can effectively handle multi-modal action distributions; for example, this can be achieved for the Soft Actor-Critic algorithm (Haarnoja et al., 2018b) using the reparametrization trick (Kingma and Ba, 2014) with normalizing flows (Rezende and Mohamed, 2015) as described in Haarnoja et al. (2018a). The final algorithm is summarized in algorithm 3.1.

3.5 Experiments

We implement the DAC algorithm described in section 3.4.4 using TensorFlow Eager (Abadi et al., 2015), and we evaluated it on popular benchmarks for continuous control simulated in MuJoCo (Todorov et al., 2012). We also define a new set of robotic continuous control tasks, which we describe in detail below, simulated in PyBullet (Coumans and Bai, 2016) and a Virtual Reality (VR) system for capturing human examples in this environment; human examples constitute a particularly challenging demonstration source due to their noisy, multi-modal and potentially sub-optimal nature, and we define episodic multi-task environments as a challenging setup for adversarial imitation learning.

We use the same architecture for the critic and policy networks as in Fujimoto

Algorithm 3.1 Discriminator Actor-Critic Adversarial Imitation Learning Algorithm

Input: expert replay buffer $\mathcal{R}_E$

procedure WRAPFORABSORBINGSTATES(τ)
  if s_T is a terminal state **then**
    $(s_T, a_T, \cdot, s'_T) \leftarrow (s_T, a_T, \cdot, s_a)$
    $\tau \leftarrow \tau \cup \{(s_a, \cdot, \cdot, s_a)\}$
  return τ
Initialize replay buffer $\mathcal{R} \leftarrow \emptyset$
for $\tau = \{(s_t, a_t, \cdot, s'_t)\}_{t=1}^{T} \in \mathcal{R}_E$ **do**
  $\tau \leftarrow$ WrapForAbsorbingState(τ) ▷ Wrap expert rollouts with absorbing states
for $n = 1, \ldots,$ **do**
  Sample $\tau = \{(s_t, a_t, \cdot, s'_t)\}_{t=1}^{T}$ with π_θ
  $\mathcal{R} \leftarrow \mathcal{R} \cup$ WrapForAbsorbingState(τ) ▷ Update Policy Replay Buffer
  for $i = 1, \ldots, |\tau|$ **do**
    $\{(s_t, a_t, \cdot, \cdot)\}_{t=1}^{B} \sim \mathcal{R}, \;\; \{(s'_t, a'_t, \cdot, \cdot)\}_{t=1}^{B} \sim \mathcal{R}_E$ ▷ Mini-batch sampling
    $\mathcal{L} = \sum_{b=1}^{B} \log D(s_b, a_b) + \log(1 - D(s'_b, a'_b))$
    Update D with GAN+GP
  for $i = 1, \ldots, |\tau|$ **do**
    $\{(s_t, a_t, \cdot, s'_t)\}_{t=1}^{B} \sim \mathcal{R}$
    for $b = 1, \ldots, B$ **do**
      $r \leftarrow \log D(s_b, a_b) - \log(1 - D(s_b, a_b))$
      $(s_b, a_b, \cdot, s'_b) \leftarrow (s_b, a_b, r, s'_b)$ ▷ Use current reward estimate.
    Update π_θ with TD3

et al. (2018a): a 2 layer MLP with ReLU activations and 400 and 300 hidden units correspondingly. We also add gradient clipping (Pascanu et al., 2013) to the actor network with a clipping value of 40. For the discriminator, we use the same architecture as in Ho and Ermon (2016): a 2 layer MLP with 100 hidden units and $\tanh$ activations. We trained all networks with the Adam optimizer (Kingma and Ba, 2014) and decay learning rate by starting with an initial learning rate of 10^{-3} and decaying it by 0.5 every 10^5 training steps for the actor network.

To make the algorithm more stable, especially in the off-policy regime when the discriminator can easily overfit to training data, we use regularization in the form of gradient penalties (Gulrajani et al., 2017) for the discriminator. Originally, this was introduced as an alternative to weight clipping for Wasserstein GANs (Arjovsky et al., 2017), but later it was shown that it helps to make JS-based GANs more stable as well (Lucic et al., 2017).

We replicate the experimental setup of Ho and Ermon (2016): expert trajectories are sub-sampled by retaining every 20 timesteps starting with a random offset (and fixed stride). Note that, as in Ho and Ermon (2016), this procedure is done to make the imitation learning task harder. When trained on original unmodified trajectories, behavioral cloning provides competitive results to GAIL.

Following Fujimoto et al. (2018a); Henderson et al. (2018), we evaluate our method using 10 different random seeds. We compute the average episode reward for each seed using 10 episodes and running the policy without exploration noise. As in Ho and Ermon (2016), we plot normalized rewards: zero corresponds to a random reward while one corresponds to expert rewards. We compute the mean over all seeds and visualize half standard deviations. In order to produce the same evaluation for GAIL, we used the

original implementation[2] of the algorithm.

3.5.1 Off Policy DAC Algorithm

We provide the DAC algorithm's evaluation results on a suite of MuJoCo tasks in fig. 3.4, as are the GAIL (TRPO) and BC baseline results. In the top-left plot, we show DAC is an order of magnitude more sample-efficient than TRPO and PPO-based GAIL baselines. In the other plots, we show that our DAC algorithm reaches comparable expected returns as the GAIL baseline using a significantly smaller number of environment steps (orders of magnitude fewer). Furthermore, DAC outperforms the GAIL baseline on all environments within a 1 million step threshold.

3.5.2 Reward Bias

As discussed in section 3.4.2, the reward function variants used with GAIL can have implicit biases when used without handling absorbing states. Figure 3.5 demonstrates how bias affects results on an environment with survival bonus when using the reward function of Ho and Ermon (2016), $r(s, a) = -\log(1-D(s, a))$. Surprisingly, when using a fixed and untrained GAIL discriminator that outputs 0.5 for every state-action pair, we could reach episode returns of around 1000 on the Hopper environment, corresponding to approximately one-third of the expert performance. Without any reward learning and using no expert demonstrations, the agent can learn a policy that outperforms behavioral cloning (see fig. 3.5). Therefore, the choice of a specific reward function might provide strong prior knowledge that helps the RL algorithm move towards recovering the expert policy, irrespective of the quality of the learned reward.

[2] https://github.com/openai/imitation

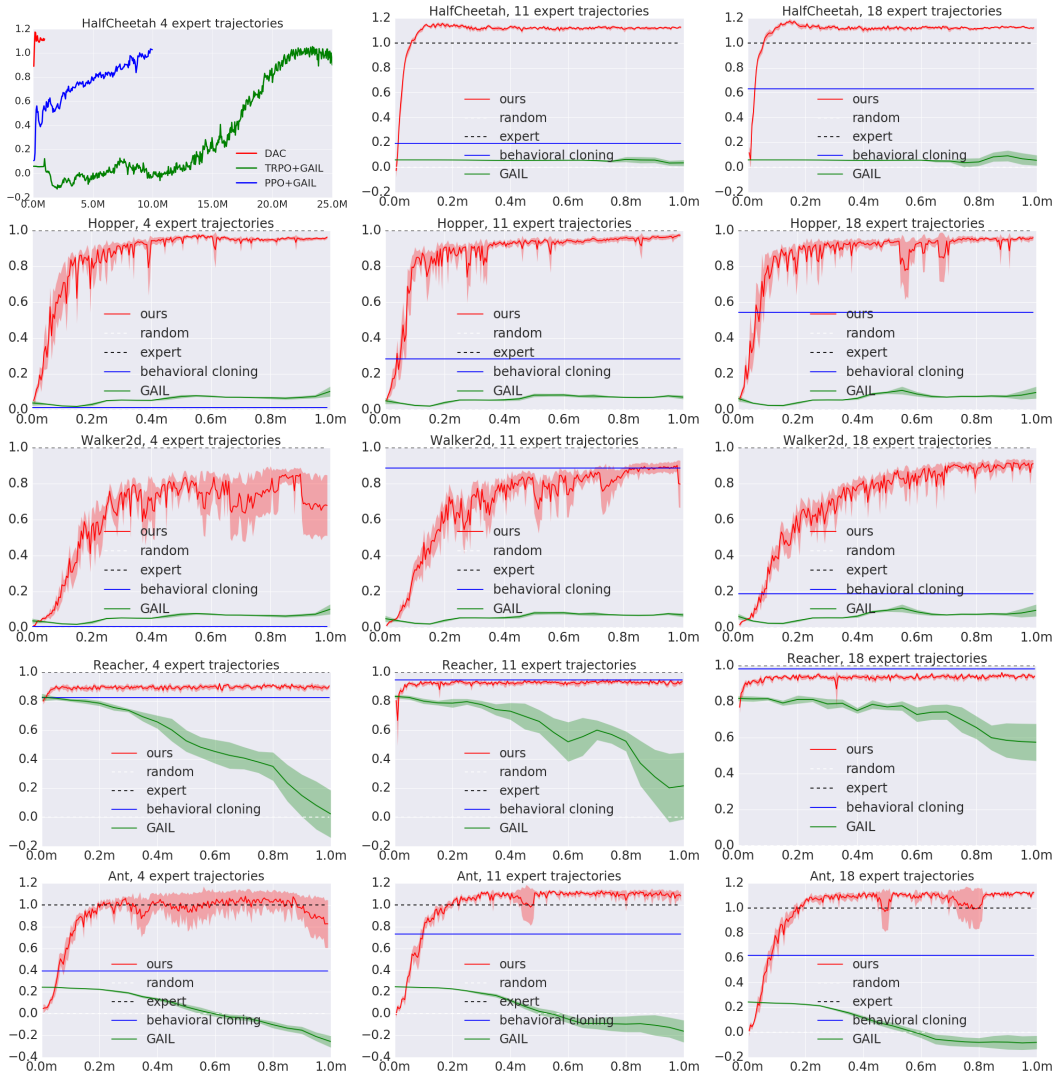


Figure 3.4: Comparisons of different algorithms given the same number of expert demonstrations. y-axis corresponds to normalized reward (0 corresponds to a random policy, while 1 corresponds to an expert policy).

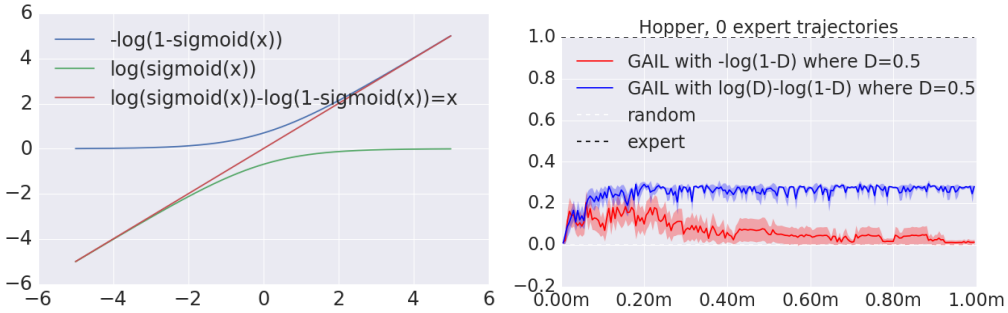


Figure 3.5: Reward functions that can be used in GAIL (left). Even without training some reward functions can perform well on some tasks (right).

Additionally, we evaluated our method on two environments with a per-step penalty (see fig. 3.7). These environments are simulated in PyBullet and consist of a Kuka IIWA arm and 3 blocks on a virtual table. A rendering of the environment can be found in fig. 3.6. Using a Cartesian displacement action for the gripper end-effector and a compact observation-space (consisting of each block's 6DOF pose and the Kuka's end-effector pose), the agent must either a) reach one of the 3 blocks in the shortest number of frames possible (the target block is provided to the policy as a one-hot vector), which we call *Kuka-Reach*, or b) push one block along the table so that it is adjacent to another block, which we call *Kuka-PushNext*. For evaluation, we define a sparse reward indicating successful task completion (within some threshold). For these imitation learning experiments, we use human demonstrations collected with a VR setup, where the participant wears a VR headset and controls in real-time the gripper end-effector using a 6DOF controller.

Using the reward defined as $r(s,a) = -log(1 - D(s,a))$ and without absorbing state handling, the agent completely fails to recover the expert policy given 600 expert trajectories without sub-sampling (as shown in fig. 3.5). In contrast, our DAC algorithm quickly learns to imitate the expert, despite using noisy and potentially sub-optimal human demonstrations.

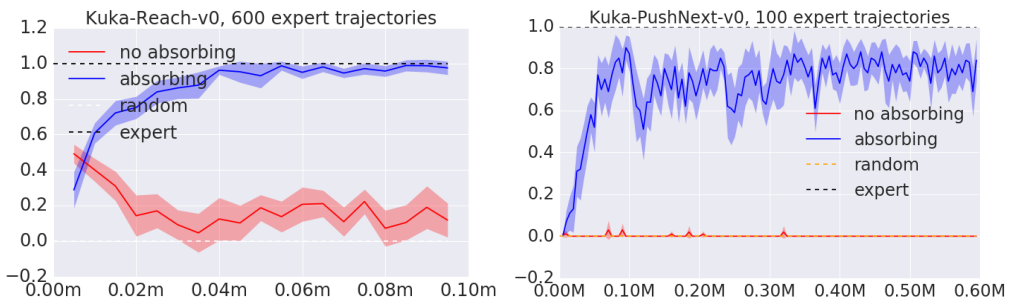


Figure 3.7: Effect of absorbing state handling on Kuka environments with human demonstrations.

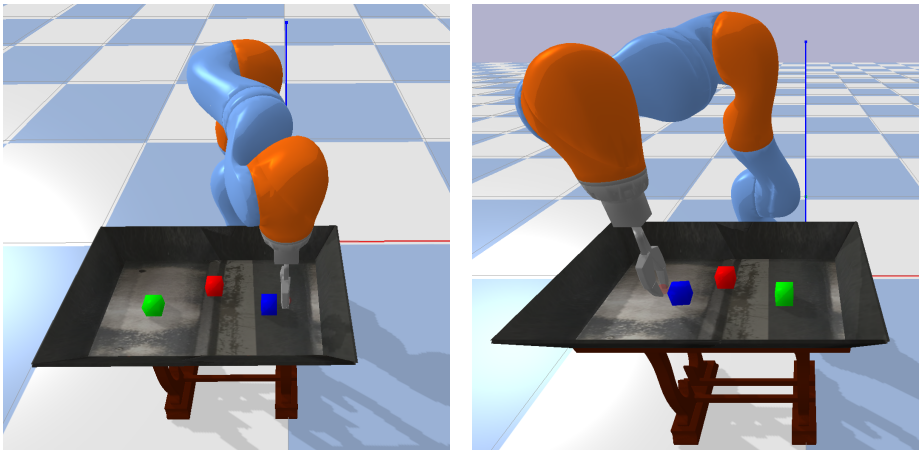


Figure 3.6: Renderings of our Kuka-IIWA environment. Using a VR headset and 6DOF controller, a human participant can control the 6DOF end-effector pose to record expert demonstrations. In the *Kuka-Reach* tasks, the agent must bring the robot gripper to 1 of the 3 blocks (where the state contains a 1-hot encoding of the task), and for the *Kuka-PushNext* tasks, the agent must use the robot gripper to push one block next to another.

As discussed, alternative reward functions do not have this positive bias but still require proper handling of the absorbing states as well in order to avoid early termination due to incorrectly assigned per-frame penalty. Figure 3.8 illustrates results for AIRL with and without learning rewards for absorbing states. For these experiments, we use the discriminator structure from Fu et al. (2017) in combination with the TD3 algorithm.

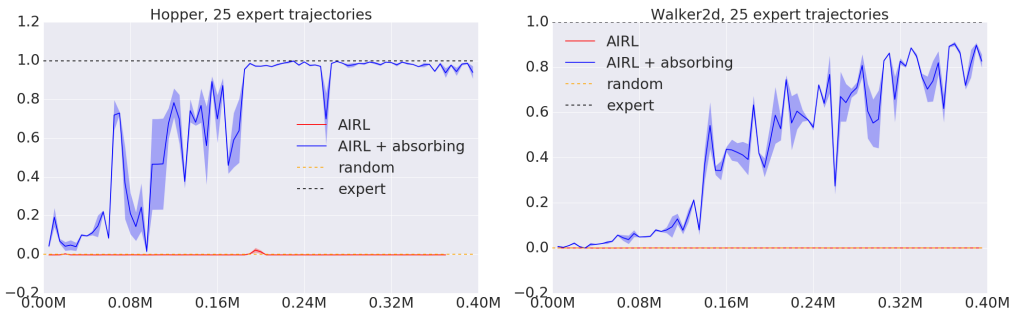


Figure 3.8: Effect of learning absorbing state rewards when using an AIRL discriminator within the DAC Framework.

3.6 Conclusion

In this work we address several important issues associated with the popular GAIL framework. In particular, we address 1) sample inefficiency with respect to policy transitions in the environment and 2) we demonstrate a number of reward biases that can either implicitly impose prior knowledge about the true reward, or alternatively, prevent the policy from imitating the optimal expert. To address reward bias, we propose a simple mechanism whereby the rewards for absorbing states are also learned, which negates the need to hand-craft a discriminator reward function for the properties of the task at hand. To improve sample efficiency, we perform off-policy training of the discriminator and use an off-policy RL algorithm. We show that our algorithm reaches state-of-the-art performance for an imitation learning algorithm on several standard RL benchmarks, and is able to recover the expert policy given a significantly smaller number of samples than in recent GAIL work.

Chapter 4

Imitation Learning via Off-Policy Distribution Matching

When performing imitation learning from expert demonstrations, distribution matching is a popular approach, in which one alternates between estimating distribution ratios and then using these ratios as rewards in a standard reinforcement learning (RL) algorithm. Traditionally, estimation of the distribution ratio requires on-policy data, which has caused previous work to either be exorbitantly data-inefficient or alter the original objective in a manner that can drastically change its optimum such as in Discriminator-Actor-Critic described in chapter 3. This work shows how the original distribution ratio estimation objective may be transformed in a principled manner to yield a completely off-policy objective. In addition to the data-efficiency that this provides, we show that this objective also renders the use of a separate RL optimization unnecessary. Rather, an imitation policy may be learned directly from this objective without the use of explicit rewards. We call the resulting algorithm *ValueDICE* and evaluate it on a suite of popular imitation learning benchmarks, finding that it can achieve state-of-the-art sample

efficiency and performance.

4.1 Introduction

Reinforcement learning (RL) is typically framed as learning a behavior policy based on reward feedback from trial-and-error experience. Accordingly, many successful demonstrations of RL often rely on carefully handcrafted rewards with various bonuses and penalties designed to encourage intended behavior (Andrychowicz et al., 2018; Nachum et al., 2019a). In contrast, many real-world behaviors are easier to demonstrate rather than devise explicit rewards. This realization is at the heart of *imitation learning* (Ho and Ermon, 2016; Ng et al., 2000; Pomerleau, 1989), in which one aims to learn a behavior policy from a set of expert demonstrations – logged experience data of a near-optimal policy interacting with the environment – without explicit knowledge of rewards.

Distribution matching via adversarial learning, or Adversarial Imitation Learning (AIL), has recently become a popular approach for imitation learning (Fu et al., 2017; Ho and Ermon, 2016; Ke et al., 2019). These methods interpret the states and actions provided in the expert demonstrations as a finite sample from a target distribution. Imitation learning can then be framed as learning a behavior policy that minimizes a divergence between this target distribution and the state-action distribution induced by the behavior policy interacting with the environment. As derived by Ho and Ermon (2016), this divergence minimization may be achieved by iteratively performing two alternating steps, reminiscent of GAN algorithms (Goodfellow et al., 2014). First, one estimates the density ratio of states and actions between the target distribution and the behavior policy. Then, these density ratios are used as rewards for a standard RL algorithm, and

the behavior policy is updated to maximize these cumulative rewards (data distribution ratios).

The main limitation of current distribution matching approaches is that estimating distribution density ratios (the first step of every iteration) typically requires samples from the behavior policy distribution. This means that every iteration – every update to the behavior policy – requires new interactions with the environment, precluding the use of these algorithms in settings where interactions with the environment are expensive and limited. Several methods attempt to relax this *on-policy* requirement and resolve the sample inefficiency problem by designing *off-policy* imitation learning algorithms, which may take advantage of past logged data, usually in the form of a replay buffer (DAC, chapter 3, and Sasaki et al. (2019)). However, these methods alter the original divergence minimization objective to measure a divergence between the target expert distribution and the replay buffer distribution. Accordingly, there is no guarantee that the learned policy will recover the desired target distribution.

In this work, we introduce an algorithm for imitation learning that, on the one hand, performs divergence minimization as in the original AIL methods, while on the other hand, is completely off-policy. We begin by providing a new formulation of the minimum divergence objective that avoids any explicit on-policy expectations. While this objective may be used in the traditional way to estimate data distribution ratios that are then input to an RL algorithm, we go further to show how the specific form of the derived objective renders using a separate RL optimization unnecessary. Rather, gradients of the minimum divergence objective with respect to behavior policy may be computed directly. This way, an imitating behavior policy may be learned to minimize the divergence without explicit rewards. We call this streamlined imitation learning algorithm ValueDICE. Besides being simpler than standard imitation learning methods, we show that our proposed algorithm

can achieve state-of-the-art performance on a suite of imitation learning benchmarks.

4.2 Background

We consider environments represented as a Markov Decision Process (MDP) (Puterman, 2014), defined by the tuple, $(\mathcal{S}, \mathcal{A}, p_0(s), p(s'|s,a), r(s,a), \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action space, respectively, $p_0(s)$ is an initial state distribution, $p(s'|s,a)$ defines environment dynamics represented as a conditional state distribution, $r(s,a)$ is a reward function, and γ is a return discount factor. A behavior policy $\pi(\cdot|\cdot)$ interacts with the environment to yield experience (s_t, a_t, r_t, s_{t+1}), for $t = 0, 1, \ldots$, where $s_0 \sim p_0(\cdot)$, $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim p(\cdot|s_t, a_t)$, $r_t = r(s_t, a_t)$. Without loss of generality, we consider infinite-horizon, non-terminating environments. In standard RL, one aims to learn a behavior policy $\pi(\cdot|s)$ to maximize cumulative rewards, based on experience gained from interacting with the environment.

In imitation learning (Abbeel and Ng, 2004; Ho and Ermon, 2016; Pomerleau, 1989), the environment reward is not observed. Rather, one has access to a set of expert demonstrations $\mathcal{D} := \{(s_k, a_k, s'_k\}_{k=1}^N$ given by state-action-next-state transitions in the environment induced by an unknown expert policy π_{exp} and the goal is to learn a behavior policy π which recovers π_{exp}. During the learning process, in addition to the finite set of expert demonstrations $\mathcal{D}$, one may also optionally interact with the environment (in these interactions, no rewards are observed). This setting describes several real-world applications where rewards are unknown, such as Bojarski et al. (2016); Muller et al. (2006); Pomerleau (1989).

4.2.1 Behavioral Cloning (BC)

Supervised behavioral cloning (BC) is a popular approach for imitation learning. Given a set of expert demonstrations, a mapping of state observations to actions is fit using regression or density estimation. In the simplest case, one simply trains the behavior policy π to minimize the negative log-likelihood of the observed expert actions:

$$\min_{\pi} J_{\mathrm{BC}}(\pi) := -\frac{1}{N}\sum_{k=1}^{N} \log \pi(a_k|s_k). \tag{4.1}$$

Unlike Inverse Reinforcement Learning (IRL) algorithms (e.g., GAIL (Ho and Ermon, 2016)), BC does not perform any additional policy interactions with the learning environment and hence does not suffer from the same issue of policy sample complexity. However, behavioral cloning suffers from distributional drift (Ross et al., 2011); i.e., there is no way for π to learn how to recover if it deviates from the expert behavior to a state $\tilde{s}$ not seen in the expert demonstrations.

4.2.2 Distribution Matching

The distribution matching approach provides a family of methods that are robust to distributional shifts. Rather than considering the policy directly as a conditional distribution $\pi(\cdot|s)$ over actions, this approach considers the state-action distribution induced by a policy. In particular, under certain conditions (Puterman, 2014), there is a one-to-one correspondence between a policy and its state-action distribution defined as

$$d^{\pi}(s,a) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}p(s_t = s, a_t = a | s_0 \sim p_0(\cdot),$$
$$s_t \sim p(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)).$$

By the same token, the unknown expert policy π_{exp} also possesses a state-action distribution d^{exp}, and one may assume that the expert demonstrations $\mathcal{D} := \{(s_k, a_k, s'_k\}_{k=1}^{N}$ are sampled as $(s_k, a_k) \sim d^{\text{exp}}, s'_k \sim p(\cdot|s_k, a_k)$.

Accordingly, the distribution matching approach proposes to learn π to minimize the divergence between d^{π} and d^{exp}. The KL-divergence is typically used to measure the discrepancy between d^{π} and d^{exp} (Ho and Ermon, 2016; Ke et al., 2019):

$$-D_{\text{KL}}\left(d^{\pi}||d^{\text{exp}}\right) = \mathop{\mathbb{E}}_{(s,a)\sim d^{\pi}}\left[\log\frac{d^{\text{exp}}(s,a)}{d^{\pi}(s,a)}\right]. \tag{4.2}$$

The use of the KL-divergence is convenient, as it may be expressed as an RL problem where rewards are given by log distribution ratios:

$$-D_{\text{KL}}\left(d^{\pi}||d^{\text{exp}}\right) = (1-\gamma)\cdot \mathop{\mathbb{E}}_{\substack{s_0\sim p_0(\cdot),\ a_t\sim\pi(\cdot|s_t)\\ s_{t+1}\sim p(\cdot|s_t,a_t)}}\left[\sum_{t=0}^{\infty}\gamma^{t}\log\frac{d^{\text{exp}}(s_t,a_t)}{d^{\pi}(s_t,a_t)}\right]. \tag{4.3}$$

In other words, if one has access to estimates of the distribution ratios of the two policies, then the minimum divergence problem reduces to a max-return RL problem with rewards $\tilde{r}(s,a) = \log\frac{d^{\text{exp}}(s,a)}{d^{\pi}(s,a)}$. Any on-policy or off-policy RL algorithm can be used to maximize the corresponding expected returns in eq. (4.3).

Capitalizing on this observation, Ho and Ermon (2016) and Ke et al. (2019) propose algorithms (e.g., GAIL) in which the distribution ratio is estimated using a GAN-like

objective:

$$\max_{h:\mathcal{S}\times\mathcal{A}\to(0,1)} J_{\mathrm{GAIL}}(h) := \mathop{\mathbb{E}}_{(s,a)\sim d^{\exp}}[\log h(s,a)] + \mathop{\mathbb{E}}_{(s,a)\sim d^{\pi}}[\log(1-h(s,a))]. \tag{4.4}$$

In this objective, the function h acts as a *discriminator*, discriminating between samples (s,a) from $d^{\exp}$ and d^{π}. The optimal discriminator satisfies,

$$\log h^*(s,a) - \log(1-h^*(s,a)) = \log \frac{d^{\exp}(s,a)}{d^{\pi}(s,a)}, \tag{4.5}$$

and so the distribution matching rewards may be computed as

$$\tilde{r}(s,a) = \log h^*(s,a) - \log(1-h^*(s,a)).$$

In practice, the discriminator is not fully optimized, and instead gradient updates to the discriminator and policy are alternated.

These prior distribution matching approaches possess two limitations which we will resolve with our proposed ValueDICE algorithm:

- **On-policy.** Arguably the main limitation of these prior approaches is that they require access to on-policy samples from d^{π}. While off-policy RL can be used for learning π, optimizing the discriminator h necessitates having on-policy samples (the second expectation in eq. (4.4)). Thus, in practice, GAIL requires a prohibitively large number of environment interactions, making it unfeasible for use in many real-world applications. Attempts to remedy this, such as Discriminator-Actor-Critic (DAC, chapter 3), often do so via ad-hoc methods; for example, changing the on-policy expectation $\mathbb{E}_{(s,a)\sim d^{\pi}}[\log(1-h(s,a))]$ in eq. (4.4) to an expectation over the replay buffer $\mathbb{E}_{(s,a)\sim d^{\mathrm{RB}}}[\log(1-h(s,a))]$. While DAC achieves good empirical results, it does not

guarantee distribution matching of π to π_{exp}, especially when d^{RB} is far from d^{π}.

- **Separate RL optimization.** Prior approaches require iteratively taking alternating steps: first estimate the data distribution ratios using the GAN-like objective, then input these into an RL optimization and repeat. A separate RL algorithm introduces complexity to any implementation of these approaches, with many additional design choices that need to be made and more function approximators to learn (e.g., value functions). Our introduced ValueDICE will be shown not to need a separate RL optimization.

4.3 Off-policy Formulation of the KL-Divergence

As is standard in distribution matching, we begin with the KL-divergence between the policy state-action occupancies and the expert. However, in contrast to the form used in eq. (4.3) or eq. (4.4), we use the Donsker-Varadhan representation (Donsker and Varadhan, 1983) given by,

$$-D_{\text{KL}}\left(d^{\pi}||d^{\text{exp}}\right) = \min_{x:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} \log \mathop{\mathbb{E}}_{(s,a)\sim d^{\text{exp}}}[e^{x(s,a)}] - \mathop{\mathbb{E}}_{(s,a)\sim d^{\pi}}[x(s,a)]. \tag{4.6}$$

Similar to eq. (4.4), this dual representation of the KL has a property that is important for imitation learning. The optimal x^* is equal to the log distribution ratio (plus a constant):[1]

$$x^*(s,a) = \log \frac{d^{\pi}(s,a)}{d^{\text{exp}}(s,a)} + C. \tag{4.7}$$

[1]This result is easy to derive by setting the gradient of the Donsker-Varadhan representation to zero and solving for x^*.

In our considered infinite-horizon setting, the constant does not affect optimality, so we will ignore it (take $C = 0$). If one were to take a GAIL-like approach, they could use this form of the KL to estimate distribution matching rewards given by $\tilde{r}(s, a) = -x^*(s, a)$, and any standard RL algorithm could then maximize these. However, there is no clear advantage of this objective over GAIL since it still relies on an expectation with respect to on-policy samples from d^π.

To make this objective practical for off-policy learning, we take inspiration from derivations used in DualDICE (Nachum et al., 2019b), and perform the following change of variables:[2]

$$x(s, a) = \nu(s, a) - \mathcal{B}^\pi \nu(s, a), \tag{4.8}$$

where $\mathcal{B}^\pi$ is the expected Bellman operator with respect to policy π and zero reward:

$$\mathcal{B}^\pi \nu(s, a) = \gamma \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a), a' \sim \pi(\cdot|s')} [\nu(s', a')]. \tag{4.9}$$

This change of variables is explicitly chosen to take advantage of the linearity of the second expectation in eq. (4.6). Specifically, the representation becomes,

$$-D_{\mathrm{KL}}\left(d^\pi || d^{\mathrm{exp}}\right) = \min_{\nu: \mathcal{S} \times \mathcal{A} \to \mathbb{R}} \log \mathop{\mathbb{E}}_{(s,a) \sim d^{\mathrm{exp}}} [e^{\nu(s,a) - \mathcal{B}^\pi \nu(s,a)}] - \mathop{\mathbb{E}}_{(s,a) \sim d^\pi} [\nu(s, a) - \mathcal{B}^\pi \nu(s, a)], \tag{4.10}$$

where the second expectation conveniently telescopes and reduces to an expectation over initial states (see Nachum et al. (2019b) for details):

$$\min_{\nu: \mathcal{S} \times \mathcal{A} \to \mathbb{R}} J_{\mathrm{DICE}}(\nu) := \log \mathop{\mathbb{E}}_{(s,a) \sim d^{\mathrm{exp}}} [e^{\nu(s,a) - \mathcal{B}^\pi \nu(s,a)}] - (1-\gamma) \cdot \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0(\cdot), \\ a_0 \sim \pi(\cdot|s_0)}} [\nu(s_0, a_0)]. \tag{4.11}$$

[2]This change of variables is valid when one assumes $\log d^\pi(s, a)/d^{\mathrm{exp}}(s, a) \in \mathcal{K}$ for all $s \in \mathcal{S}, a \in \mathcal{A}$, where $\mathcal{K}$ is some bounded subset of $\mathbb{R}$, and x is restricted to the family of functions $\mathcal{S} \times \mathcal{A} \to \mathcal{K}$.

Thus we achieve our *ValueDICE*[3] objective. It allows us to express the KL-divergence between d^π and d^{exp} in terms of an objective over a 'value function ν expressed in an off-policy manner, with expectations over expert demonstrations d^{exp} and initial state distribution $p_0(\cdot)$.

It is clear that the derived objective in eq. (4.11) possesses two key characteristics missing from prior distribution matching algorithms: First, the objective does not rely on access to samples from the on-policy distribution d^π, and so may be used in more realistic, off-policy settings. Second, the objective describes a proper divergence between d^π and d^{exp}, instead of estimating a divergence between d^{RB} and d^{exp}, and thus avoids poor behavior when d^{RB} is far from d^π. In the following section, we will go further to show how the objective in eq. (4.11) also renders the use of a separate RL optimization unnecessary.

4.4 Imitation Learning with Implicit Rewards

Although it is standard in distribution matching to have separate optimizations for estimating the distribution ratios and learning a policy, in our case, this can be mitigated. Indeed, looking at our formulation of the KL in eq. (4.11), we see that gradients of this objective with respect to π may be easily computed. Specifically, we may express the distribution matching objective for π as a $\max$-$\min$ optimization:

$$\max_{\pi} \min_{\nu:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} J_{\text{DICE}}(\pi,\nu) := \log \mathop{\mathbb{E}}_{(s,a)\sim d^{\text{exp}}}\left[e^{\nu(s,a)-\mathcal{B}^\pi\nu(s,a)}\right] - (1-\gamma)\cdot \mathop{\mathbb{E}}_{\substack{s_0\sim p_0(\cdot),\\ a_0\sim\pi(\cdot|s_0)}}\left[\nu(s_0,a_0)\right]. \tag{4.12}$$

[3]DICE (Nachum et al., 2019b) is an abbreviation for discounted distribution correction estimation.

If the inner objective over ν is sufficiently optimized, the gradients of π may be computed directly (Bertsekas, 1999), noting that,

$$\frac{\partial}{\partial \pi} e^{\nu(s,a) - \mathcal{B}^\pi \nu(s,a)} = -\gamma \cdot e^{\nu(s,a) - \mathcal{B}^\pi \nu(s,a)} \cdot \mathop{\mathbb{E}}_{s' \sim T(s,a), a' \sim \pi(s')} [\nu(s', a') \nabla \log \pi(a'|s')], \quad (4.13)$$

$$\frac{\partial}{\partial \pi} \mathop{\mathbb{E}}_{s_0 \sim p_0(\cdot), a_0 \sim \pi(\cdot|s_0)} [\nu(s_0, a_0)] = \mathop{\mathbb{E}}_{s_0 \sim p_0(\cdot), a_0 \sim \pi(\cdot|s_0)} [\nu(s_0, a_0) \nabla \log \pi(a_0|s_0)]. \quad (4.14)$$

In continuous control environments when π is parameterized by a Gaussian and ν is a neural network, one may use the re-parameterization trick (Haarnoja et al., 2018b) to compute gradients of the ν-values for policy mean and variance directly as opposed to computing $\nabla \log \pi(a|s)$. Please see the appendix for a full pseudocode implementation of ValueDICE. We note that in practice, as in GAIL, we do not train ν until optimality but rather alternate ν and π updates.

The mechanics of learning π according to the ValueDICE objective are straightforward, but what is the underlying reason for this more streamlined policy learning? How does it relate the standard protocol of alternating data distribution estimation with RL optimization? To better understand this, we consider the form of ν when it is completely optimized. If we consider the original change of variables (eq. (4.8)) and optimality characterization (eq. (4.7)) we have,

$$\nu^*(s,a) - \mathcal{B}^\pi \nu^*(s,a) = x^*(s,a) = \log \frac{d^\pi(s,a)}{d^{\exp}(s,a)}. \quad (4.15)$$

From this characterization of ν^*, we realize that ν^* is a sort of Q-value function: $\nu^*(s,a)$ is the future discounted sum of rewards $\tilde{r}(s,a) := \log \frac{d^\pi(s,a)}{d^{\exp}(s,a)}$ when acting according to π. The gradients for π then encourage the policy to choose actions which *minimize* $\nu^*(s,a)$, i.e., *maximize* future discounted log ratios $\log \frac{d^{\exp}(s,a)}{d^\pi(s,a)}$. Thus we realize that the objective

for π in ValueDICE performs exactly the RL optimization suggested by eq. (4.3). The streamlined nature of ValueDICE comes from the fact that the value function ν (which would traditionally need to be learned as a critic in a separate actor-critic RL algorithm) is learned directly from the same objective as that used for distribution matching.

Thus, in addition to estimating a proper divergence between d^π and d^{exp} in an off-policy manner, ValueDICE also greatly simplifies the implementation of distribution matching algorithms. There is no longer a need to use a separate RL algorithm for learning π. The use of ν as a value function removes any use of explicit rewards. Instead, the objective and implementation are only in terms of policy π and function ν.

4.5 Some Practical Considerations

In order to make use of the ValueDICE objective (eq. (4.12)) in practical scenarios, where one does not have access to d^{exp} or $p_0(\cdot)$ but rather only limited, finite samples, we perform several modifications.

Empirical Expectations The objective in eq. (4.12) contains three expectations:

1. An expectation over d^{exp} (the first term of the objective). Note that this expectation has a logarithm outside of it, which would make any mini-batch approximations of the gradient of this expectation biased.

2. An expectation over $p_0(\cdot)$ (the second term of the objective). This term is linear and so is very amenable to mini-batch optimization.

3. An expectation over the environment transition $p(\cdot|s, a)$ used to compute $\mathcal{B}^\pi\nu(s, a)$. This expectation has a log-expected-exponent applied to it, so its mini-batch approximated gradient would be biased in general.

For the first expectation, previous works have suggested several remedies to reduce the bias of mini-batch gradients, such as maintaining moving averages of various quantities (Belghazi et al., 2018). In the setting, we considered, we found this to have a negligible effect on performance. In fact, simply using the biased mini-batched gradients was sufficient for good performance, and so we used this for our experiments.

For the second expectation, we use standard mini-batch gradients, which are unbiased. Although initial state distributions are usually not used in imitation learning, it is easy to record initial states as observed and thus have access to an empirical sample from p_0. Furthermore, as detailed in section 4.5, it is possible to modify the initial state distribution used in the objective without adverse effects.

Finally, for the third expectation, previous works have suggested the use of Fenchel conjugates to remove the bias (Nachum et al., 2019b). In our case, we found this unnecessary and instead use a biased estimate based on the single sample $s' \sim p(\cdot|s,a)$. This naive approach was enough to achieve good performance on the benchmark domains we considered.

In summary, the empirical form of the objective is given by,

$$\hat{J}_{\mathrm{DICE}}(\pi,\nu) = \mathop{\mathbb{E}}_{\substack{\mathrm{batch}(\mathcal{D})\sim\mathcal{D},\\ \mathrm{batch}(p_0)\sim\hat{p}_0}} \left[\log \mathop{\mathbb{E}}_{\substack{s,a,s'\sim\mathrm{batch}(\mathcal{D}),\\ a'\sim\pi(\cdot|s')}} \left[e^{\nu(s,a)-\gamma\nu(s',a')} \right] - (1-\gamma)\cdot \mathop{\mathbb{E}}_{\substack{s_0\sim\mathrm{batch}(p_0),\\ a_0\sim\pi(\cdot|s_0)}} \left[\nu(s_0,a_0)\right] \right], \tag{4.16}$$

where $\mathrm{batch}(\mathcal{D})$ is a mini-batch from $\mathcal{D}$ and $\mathrm{batch}(p_0)$ is a mini-batch from the recorded initial states $\hat{p}_0$.

Replay Buffer Regularization The original ValueDICE objective uses only expert samples and the initial state distribution. In practice, the number of expert samples may be small and lack diversity, hampering learning. In order to increase the diversity of samples used for training, we consider an alternative objective, with a controllable regularization based on experience in the replay buffer:

$$J^{\text{mix}}_{\text{DICE}}(\pi, \nu) := \log \mathop{\mathbb{E}}_{(s,a)\sim d^{\text{mix}}}[e^{\nu(s,a)-\mathcal{B}^\pi\nu(s,a)}] - (1-\alpha)(1-\gamma)\cdot \mathop{\mathbb{E}}_{\substack{s_0\sim p_0(\cdot),\\ a_0\sim\pi(\cdot|s_0)}}[\nu(s_0,a_0)] - \alpha \mathop{\mathbb{E}}_{(s,a)\sim d^{\text{RB}}}[\nu(s,a) - \mathcal{B}^\pi\nu(s,a)], \quad (4.17)$$

where $d^{\text{mix}}(s,a) = (1-\alpha)d^{\text{exp}}(s,a) + \alpha d^{\text{RB}}(s,a)$.

The main advantage of this formulation is that it introduces ν-values into the objective on samples that are outside the given expert demonstrations. Thus, if π deviates from the expert trajectory, we will still be able to learn optimal actions that return the policy back to the expert behavior. At the same time, one can verify that in this formulation the optimal π still matches π_{exp}, unlike other proposals for incorporating a replay buffer distribution (DAC, chapter 3). Indeed, the objective in eq. (4.17) corresponds to the Donsker-Varadhan representation,

$$-D_{\text{KL}}((1-\alpha)d^\pi + \alpha d^{\text{RB}} \,||\, (1-\alpha)d^{\text{exp}} + \alpha d^{\text{RB}}) = \min_{x:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} \log \mathop{\mathbb{E}}_{(s,a)\sim d^{\text{mix}}}[e^{x(s,a)}] - (1-\alpha)\cdot \mathop{\mathbb{E}}_{(s,a)\sim d^\pi}[x(s,a)] - \alpha\cdot \mathop{\mathbb{E}}_{(s,a)\sim d^{\text{RB}}}[x(s,a)], \quad (4.18)$$

and so the optimal values of ν^* satisfy,

$$\nu^*(s,a) - \mathcal{B}^\pi\nu^*(s,a) = x^*(s,a) = \log \frac{(1-\alpha)d^\pi(s,a) + \alpha d^{\text{RB}}(s,a)}{(1-\alpha)d^{\text{exp}}(s,a) + \alpha d^{\text{RB}}(s,a)}. \quad (4.19)$$

Therefore, the global optimality of $\pi = \pi_{\text{exp}}$ is unaffected by any choice of $\alpha < 1$. We note that in practice, we use a small value $\alpha = 0.1$ for regularization.

Initial State Sampling Recall that d^{exp}, d^{π} traditionally refer to *discounted* state-action distributions. That is, sampling from them is equivalent to first sampling a trajectory $(s_0, a_0, s_1, a_1, \ldots, s_T)$ and then sampling a time index t from a geometric distribution $\text{Geom}(1-\gamma)$ (appropriately handling samples that are beyond T). This means that samples far into the trajectory do not contribute much to the objective. To remedy this, we propose treating every state in a trajectory as an 'initial state.' That is, we consider a single environment trajectory $(s_0, a_0, s_1, a_1, \ldots, s_T)$ as T distinct *virtual* trajectories $\{(s_t, a_t, s_{t+1}, a_{t+1}, \ldots, s_T)\}_{t=0}^{T-1}$. We apply this to both d^{exp} and d^{π}, so that not only does it increase the diversity of samples from d^{exp}, but it also expands the initial state distribution $p_0(\cdot)$ to encompass every state in a trajectory. We note that this does not affect the optimality of the objective with respect to π, since in Markovian environments, an expert policy should be expert regardless of the state at which it starts (Puterman, 2014).

We present pseudocode for the final imitation learning algorithm that incorporates all techniques described in this section in algorithm 4.1.

Algorithm 4.1 ValueDICE

Input: expert replay buffer $\mathcal{R}_E$

Initialize replay buffer $\mathcal{R} \leftarrow \emptyset$

for $n = 1, \ldots,$ **do**

Sample (s, a, s') with π_θ

Add (s, a, s') to the replay buffer $\mathcal{R}$

$\{(s^{(i)}, a^{(i)}, s'^{(i)})\}_{i=1}^{B} \sim \mathcal{R}$ ▷ Geometric sampling

$\{(s_0^{(i)}, s_E^{(i)}, a_E^{(i)}, s_E'^{(i)})\}_{i=1}^{B} \sim \mathcal{R}_E$ ▷ Geometric sampling, $s_0^{(i)}$ is a starting episode state for $s_E^{(i)}$

$a_0^{(i)} \sim \pi_\theta(\cdot|s_0^{(i)})$, for $i = 1, \ldots, B$

$a'^{(i)} \sim \pi_\theta(\cdot|s'^{(i)})$, for $i = 1, \ldots, B$

$a_E'^{(i)} \sim \pi_\theta(\cdot|s_E'^{(i)})$, for $i = 1, \ldots, B$

Compute loss on expert data:

$\hat{J}_{log} = \log(\frac{1}{B}\sum_{i=1}^{B}((1-\alpha)e^{\nu_\psi(s_E^{(i)}, a_E^{(i)}) - \gamma\nu_\psi(s_E'^{(i)}, a_E'^{(i)})} + \alpha e^{\nu_\psi(s^{(i)}, a^{(i)}) - \gamma\nu_\psi(s'^{(i)}, a'^{(i)})}))$

Compute loss on the replay buffer:

$\hat{J}_{linear} = \frac{1}{B}\sum_{i=1}^{B}((1-\alpha)(1-\gamma)\nu_\psi(s_0^{(i)}, a_0^{(i)}) + \alpha(\nu_\psi(s^{(i)}, a^{(i)}) - \gamma\nu_\psi(s'^{(i)}, a'^{(i)})))$

Update $\psi \leftarrow \psi - \eta_\nu \nabla_\psi(\hat{J}_{log} - \hat{J}_{linear})$

Update $\theta \leftarrow \psi + \eta_\pi \nabla_\theta(\hat{J}_{log} - \hat{J}_{linear})$

4.6 Related Work

In recent years, the development of Adversarial Imitation Learning has been mostly focused on on-policy algorithms. After Ho and Ermon (2016) proposed GAIL to perform imitation learning via adversarial training, many extensions have been introduced. Many of these applications of the AIL framework (Fu et al., 2017; Hausman et al., 2017; Li et al., 2017) maintain the same form of distribution ratio estimation as GAIL necessitates on-policy samples. In contrast, our work presents an off-policy formulation of the same objective.

Although several works have attempted to apply the AIL framework to off-policy settings, these previous approaches are markedly different from our own. For example, we propose to train the discriminator in the GAN-like AIL objective using samples from a

replay buffer instead of samples from a policy in chapter 3. This changes the distribution ratio estimation to measure a divergence between the expert and the replay. Although we introduce a controllable parameter α for incorporating samples from the replay buffer into the data distribution objective, we note that in practice, we use a very small $\alpha = 0.1$. Furthermore, by using samples from the replay buffer in *both* terms of the objective instead of just one, the global optimality of the expert policy is not affected.

The off-policy formulation of the KL-divergence we derive is motivated by similar techniques in DualDICE (Nachum et al., 2019b). Still, our use of these techniques provides several novelties. First, Nachum et al. (2019b) only use the divergence formulation for data distribution estimation (which is used for off-policy evaluation), assuming a fixed policy. We use the formulation for learning a policy to minimize the divergence directly. Moreover, previous works have only applied these derivations to the f-divergence form of the KL-divergence, while we are the first to utilize the Donsker-Varadhan form. Anecdotally in our initial experiments, we found that using the f-divergence form leads to poor performance. We note that our proposed objective follows a form similar to REPS (Peters et al., 2010), which also utilizes a log-average-exp term. However, policy and value learning in REPS are performed via a bi-level optimization (i.e., the policy is learned with respect to a different objective), which is distinct from our algorithm, which trains values and policy with respect to the same objective. Our proposed ValueDICE is also significant for incorporating arbitrary (non-expert) data into its learning.

4.7 Experiments

We evaluate ValueDICE in various settings, starting with a simple synthetic task before continuing to an evaluation on a suite of MuJoCo benchmarks.

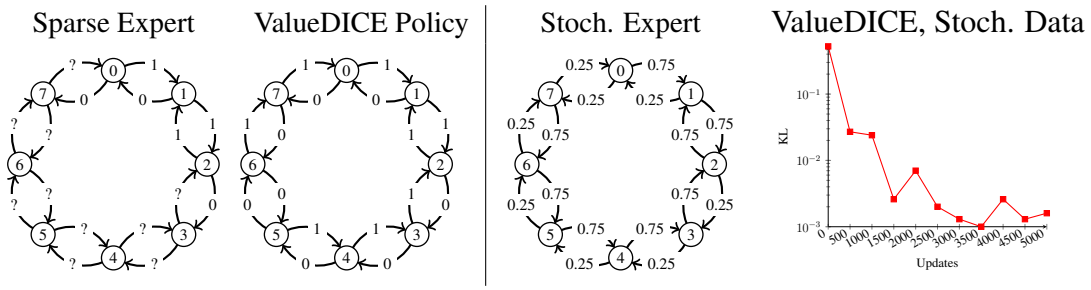


Figure 4.1: Results of ValueDICE on a simple Ring MDP. **Left**: The expert data is sparse and only covers states 0, 1, and 2. Nevertheless, ValueDICE is able to learn a policy on *all* states to best match the observed expert state-action occupancies (the policy learns to always go to states 1 and 2). **Right**: The expert is stochastic. ValueDICE is able to learn a policy which successfully minimizes the true KL computed between d^{π} and d^{exp}.

4.7.1 Implementation Details

All algorithms use networks with an MLP architecture with 2 hidden layers and 256 hidden units. For discriminators, critic, ν we use Adam optimizer with learning rate 10^{-3} while for the actors, we use the learning rate of 10^{-5}. For the discriminator and ν networks we use gradient penalties from Gulrajani et al. (2017). We also regularize the actor network with the orthogonal regularization (Brock et al., 2018) with a coefficient 10^{-4}. Also, we perform 4 updates per 1 environment step. We handle absorbing states of the environments as discussed in chapter 3.

4.7.2 Ring MDP

We begin by analyzing the behavior of ValueDICE on a simple synthetic MDP (see fig. 4.1). The states of the MDP are organized in a ring. At each state, two actions are possible: move clockwise or counterclockwise. We first look at the performance of ValueDICE in a situation where the expert data is sparse and does not cover all states and actions. Specifically, we provide expert demonstrations which cover only states 0, 1, and 2 (see fig. 4.1 left). While the problem of recovering the true (unknown) expert is

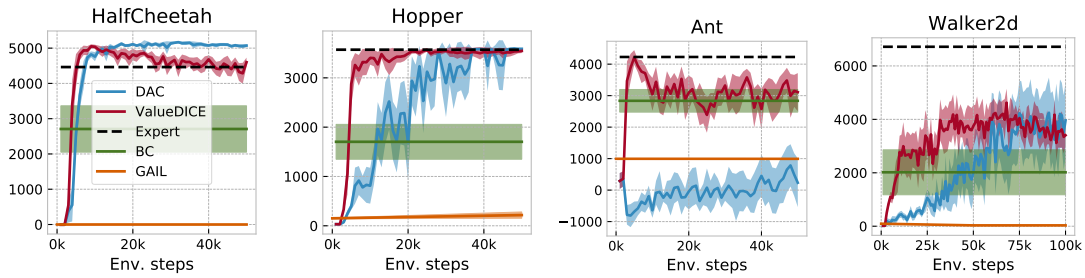


Figure 4.2: Comparison of algorithms given 1 expert trajectory. We use the original implementation of GAIL (Ho and Ermon, 2016) to produce GAIL and BC results.

ill-defined, it is still possible to find a policy that recovers close to the same occupancies. Indeed, this is the policy found by ValueDICE, which chooses the appropriate actions at each state to optimally reach states 1 and 2 (and alternating between states 1 and 2 when at these states). In many practical scenarios, this is the desired outcome – if the imitating policy somehow encounters a situation that deviates from the expert demonstrations, we would like it to return to the expert behavior as fast as possible. Notably, a technique like behavioral cloning would fail to learn this optimal policy since its learning is only based on observed expert data.

We also analyzed the behavior of ValueDICE with a stochastic expert (see fig. 4.1 right). Using a synthetic MDP, we can measure the divergence $D_{\mathrm{KL}}(d^{\pi}||d^{\mathrm{exp}})$ during training. As expected, we find that this divergence decreases during ValueDICE training.

4.7.3 MuJoCo Benchmarks

We compare ValueDICE against Discriminator-Actor-Critic (DAC), which we discuss in chapter 3, which is the state-of-the-art in sample-efficient adversarial imitation learning, as well as GAIL (Ho and Ermon, 2016). We evaluate the algorithms on the standard MuJoCo environments using expert demonstrations from Ho and Ermon (2016). We plot the average returns for the learned policies (using a mean action for sampling) every

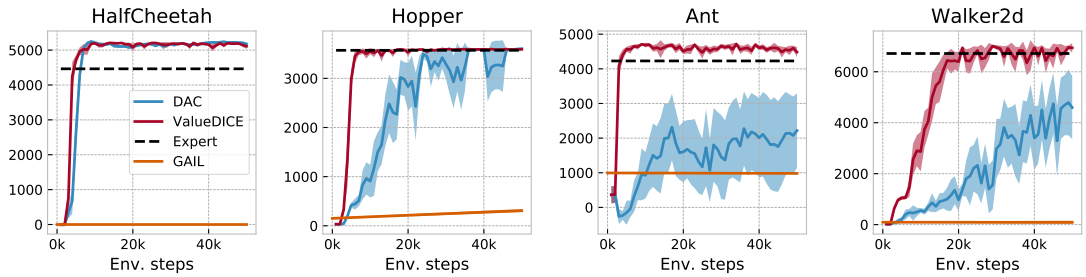


Figure 4.3: Comparison of algorithms given 10 expert trajectories. ValueDICE outperforms other methods. However, given this amount of data, BC can recover the expert policy as well.

1000 environment steps using 10 episodes. We perform this procedure for 10 different seeds and compute means and standard deviations (see fig. 4.2 and fig. 4.3, we visualize a half of standard deviation on these plots).

We present the extremely low-data regime first. In fig. 4.2 we present the imitation learning algorithms' results given only a single expert trajectory. We find that ValueDICE performs similar or better than DAC on all tasks, except Walker2d, where it converges to a slightly worse policy. Notably, in this low-data regime, behavioral cloning (BC) usually cannot recover the expert policy. We also present these algorithms' results on a larger number of expert demonstrations (see fig. 4.3). We continue to observe the strong performance of ValueDICE as well as faster convergence on all tasks. It is worth mentioning that in this large-data regime, Behavior Cloning can recover the expert performance. In all of these scenarios, GAIL is too sample-inefficient to make any progress.

We also compared ValueDICE with behavioral cloning in the offline regime when we sample no additional transitions from the learning environment (see fig. 4.4). Even given only offline data, ValueDICE outperforms behavioral cloning. For behavioral cloning, we used the same regularization as for actor training in ValueDICE.

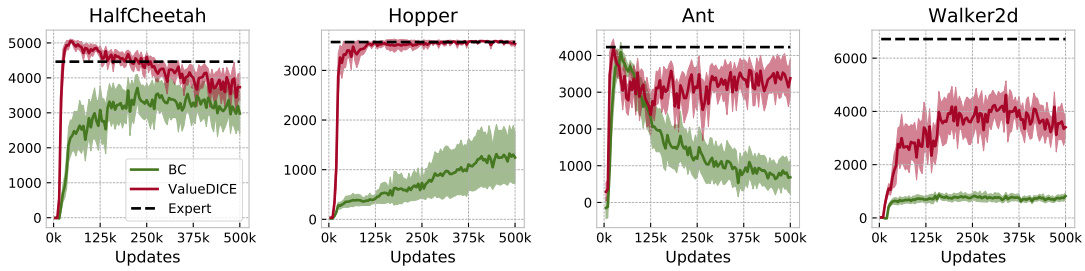


Figure 4.4: ValueDICE outperforms behavioral cloning given 1 trajectory even without replay regularization.

4.8 Conclusion

We introduced ValueDICE, an algorithm for imitation learning that outperforms the state-of-the-art on standard MuJoCo tasks. In contrast to other algorithms for off-policy imitation learning, the algorithm introduced in this paper performs robust divergence minimization in a principled off-policy manner and a strong theoretical framework. To the best of our knowledge, this is also the first algorithm for adversarial imitation learning that omits learning or defining rewards explicitly and learns a Q-function in the distribution ratio objective directly. We demonstrate the robustness of ValueDICE in a challenging synthetic tabular MDP environment and standard MuJoCo continuous control benchmark environments, and we show increased performance over baselines in both the low and high data regimes.

Chapter 5

Regularizing Deep Reinforcement Learning from Pixels

We propose a simple data augmentation technique applied to standard model-free reinforcement learning algorithms, enabling robust learning directly from pixels without the need for auxiliary losses or pre-training. The approach leverages input perturbations commonly used in computer vision tasks to transform input examples and regularize the value function and policy. Existing model-free approaches, such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018b), cannot train deep networks effectively from image pixels. However, the addition of our augmentation method dramatically improves SAC's performance, enabling it to reach state-of-the-art performance on the DeepMind control suite, surpassing model-based (Hafner et al., 2019, 2018; Lee et al., 2019) methods and recently proposed contrastive learning (Srinivas et al., 2020). Our approach, which we dub **DrQ**: **D**ata-**r**egularized **Q**, can be combined with any model-free reinforcement learning algorithm. We further demonstrate this by applying it to DQN (Mnih et al., 2013) and significantly improve its data efficiency on the Atari 100k (Kaiser et al., 2019)

benchmark.

5.1 Introduction

Sample-efficient deep reinforcement learning (RL) algorithms capable of directly training from image pixels would open up many real-world applications in control and robotics. However, simultaneously training a convolutional encoder alongside a policy network is challenging when given limited environment interaction, a strong correlation between samples, and a typically sparse reward signal. Naive attempts to use a large capacity encoder result in severe over-fitting (see fig. 5.1a), and smaller encoders produce impoverished representations that limit task performance.

Limited supervision is a common problem across AI, and several approaches are adopted:

i pre-training with self-supervised learning (SSL), followed by standard supervised learning;

ii supervised learning with an additional auxiliary loss;

iii supervised learning with data augmentation.

SSL approaches are highly effective in the large data regime, e.g., in domains such as vision (Chen et al., 2020; He et al., 2019) where large (unlabeled) datasets are readily available. However, in sample-efficient RL, training data is more limited due to restricted interaction between the agent and the environment, resulting in only 10^4–10^5 transitions from a few hundred trajectories. While there are concurrent efforts exploring SSL in the RL context (Srinivas et al., 2020), in this paper, we take a different approach, focusing on data augmentation.

A wide range of auxiliary loss functions has been proposed to augment supervised objectives, e.g., weight regularization, noise injection (Hinton et al., 2012b), or various forms of auto-encoder (Kingma et al., 2014). In RL, reconstruction objectives (Jaderberg et al., 2017; Yarats et al., 2019) or alternate tasks are often used (Dwibedi et al., 2018). However, these objectives are unrelated to the task at hand, thus have no guarantee of inducing an appropriate representation for the policy network.

Data augmentation methods have proven highly effective in vision and speech domains, where they apply output-invariant perturbations to the labeled input examples. Surprisingly, data augmentation has received relatively little attention in the RL community, which is the focus of this paper. The key idea is to use standard image transformations to perturb input observations and regularize the Q-function learned by the critic so that different transformations of the same input image have similar Q-function values. No further modifications to standard actor-critic algorithms are required, obviating the need for additional losses, e.g. based on auto-encoders (Yarats et al., 2019), dynamics models (Hafner et al., 2019, 2018), or contrastive loss terms (Srinivas et al., 2020).

The paper makes the following contributions:

i We demonstrate how straightforward image augmentation, applied to pixel observations, significantly reduces over-fitting in sample-efficient RL settings without requiring any change to the underlying RL algorithm.

ii Exploiting the MDP structure, we introduce two simple mechanisms for regularizing the value function, which are generally applicable in the context of model-free off-policy RL.

iii Combined with vanilla SAC (Haarnoja et al., 2018b) and using hyper-parameters fixed across all tasks, the overall approach obtains state-of-the-art performance on

the DeepMind control suite (Tassa et al., 2018).

iv Combined with a DQN-like agent, the approach also obtains state-of-the-art performance on the Atari 100k benchmark.

v It is thus the first practical approach that can train directly from pixels without the need for unsupervised auxiliary losses or a world model.

vi We also provide a PyTorch implementation of the approach combined with SAC and DQN.

5.2 Background

Reinforcement Learning from Images We formulate image-based control as an infinite-horizon partially observable Markov decision process (POMDP) (Bellman, 1957; Kaelbling et al., 1998). An POMDP can be described as the tuple $(\mathcal{O}, \mathcal{A}, p(\cdot|o,a), p_0(\cdot), r(o,a), \gamma)$, where $\mathcal{O}$ is the high-dimensional observation space (image pixels), $\mathcal{A}$ is the action space, the transition dynamics $p(\cdot|o_{\leq t}, a_t)$ capture the probability distribution over the next observation given the history of previous observations $o_{\leq t}$ and current action a_t, $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ is the reward function that maps the current observation and action to a reward, and $\gamma \in [0, 1)$ is a discount factor. Per common practice (Mnih et al., 2013), throughout the paper the POMDP is converted into an MDP (Bellman, 1957) by stacking several consecutive image observations into a state $s_t = \{o_t, o_{t-1}, o_{t-2}, \ldots\}$. For simplicity we redefine the transition dynamics as $p(\cdot|s,a)$ and the reward function as $r(s,a)$. We then aim to find a policy π that maximizes the cumulative discounted return

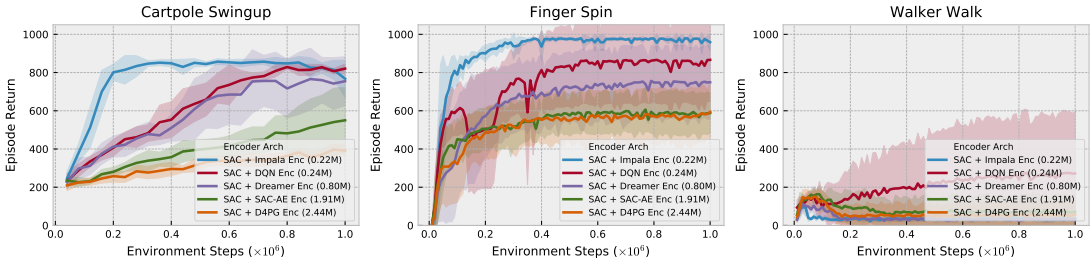
$$\mathop{\mathbb{E}}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t), s_0 \sim p_0(\cdot)].$$

Soft Actor-Critic The Soft Actor-Critic (SAC) (Haarnoja et al., 2018b) learns a state-action value function Q_θ, a stochastic policy π_θ and a temperature α to find an optimal policy for an MDP $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ by optimizing a γ-discounted maximum-entropy objective (Ziebart et al., 2008). θ generically denotes the parameters updated through training in each part of the model.

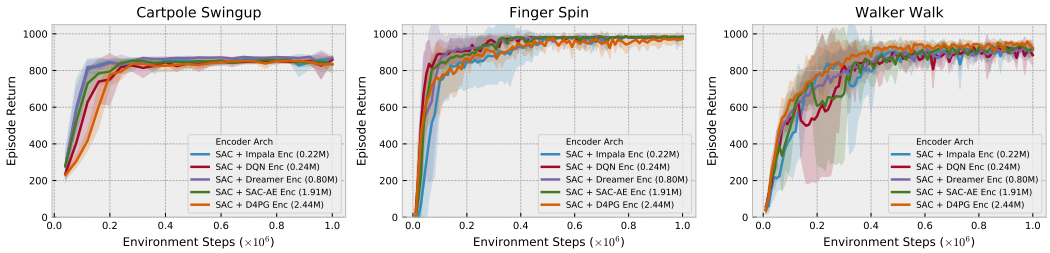
Deep Q-learning DQN (Mnih et al., 2013) also learns a convolutional neural net to approximate Q-function over states and actions. The main difference is that DQN operates on discrete action spaces; thus, we can infer the policy directly from Q-values. In practice, the standard version of DQN is frequently combined with a set of refinements that improve performance and training stability, commonly known as Rainbow (van Hasselt et al., 2015). For simplicity, the rest of the paper describes a generic actor-critic algorithm rather than DQN or SAC in particular.

5.3 Sample Efficient Reinforcement Learning from Pixels

This work focuses on the data-efficient regime, seeking to optimize performance given limited environment interaction. In fig. 5.1a we show a motivating experiment that demonstrates over-fitting to be a significant issue in this scenario. Using three tasks from the DeepMind control suite (Tassa et al., 2018), SAC (Haarnoja et al., 2018b) is trained with the same policy network architecture but using different image encoder architectures, taken from the following RL approaches: NatureDQN (Mnih et al., 2013), Dreamer (Hafner et al., 2019), Impala (Espeholt et al., 2018), SAC-AE (Yarats et al., 2019) (also used in CURL (Srinivas et al., 2020)), and D4PG (Barth-Maron et al., 2018). The encoders vary significantly in their capacity, with parameter counts ranging from



(a) Unmodified SAC.



(b) SAC with image shift augmentation.

Figure 5.1: The performance of SAC trained from pixels on the DeepMind control suite using image encoder networks of different capacity (network architectures taken from recent RL algorithms, with parameter count indicated). **(a)**: unmodified SAC. Task performance can be seen to get *worse* as the capacity of the encoder increases, indicating over-fitting. For Walker Walk (right), all architectures provide mediocre performance, demonstrating the inability of SAC to train directly from pixels on harder problems. **(b)**: SAC combined with image augmentation in the form of random shifts. The task performance is now similar for all architectures, regardless of their capacity. There is also a clear performance improvement relative to (a), particularly for the more challenging Walker Walk task.

220k to 2.4M. The curves show that *performance decreases as parameter count increases*, a clear indication of over-fitting.

5.3.1 Image Augmentation

Ciregan et al. (2012),Ciresan et al. (2011), Krizhevsky et al. (2012) and Chen et al. (2020) developed a range of successful image augmentation techniques to counter over-fitting. These apply transformations to the input image for which the task labels are invariant, e.g., for object recognition tasks, image flips and rotations do not alter the

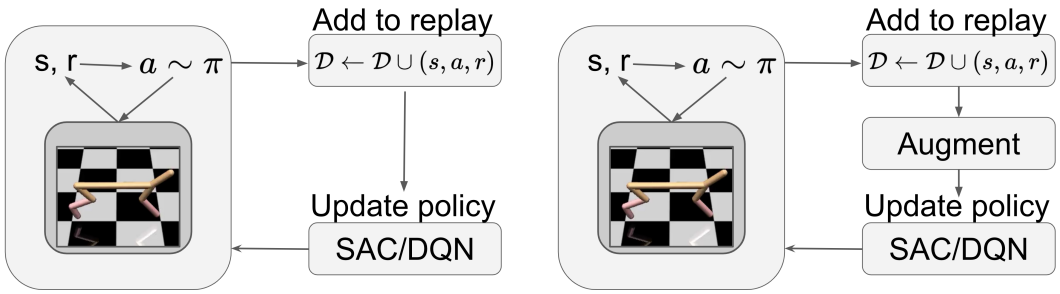


Figure 5.2: We augment standard off-policy RL algorithms with data augmentation by perturbing observations samples from the replay buffer for learning.

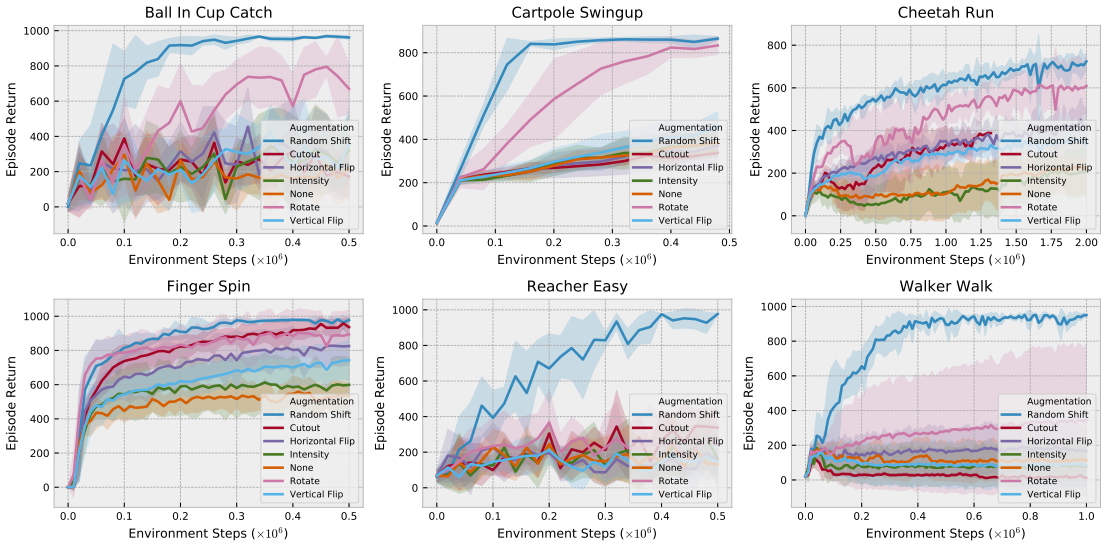


Figure 5.3: Various image augmentations have different effect on the agent's performance. Overall, we conclude that using image augmentations helps to fight overfitting. Moreover, we notice that random shifts proven to be the most effective technique for tasks from the DeepMind control suite.

semantic label (. However, RL tasks differ significantly from those in vision, and in many cases, these transformations would not preserve the reward. We examine several standard image transformations from (Chen et al., 2020) in fig. 5.3 and conclude that random shifts strike a good balance between simplicity and performance. We, therefore, limit our choice of augmentation to this transformation.

Figure 5.1b shows the results of this augmentation applied during SAC training. We apply data augmentation only to the images sampled from the replay buffer and not for the initial sample collection procedure. The images from the DeepMind control suite are 84×84. We pad each side by 4 pixels (by repeating boundary pixels) and then select a random 84×84 crop, yielding the original image shifted by ± 4 pixels. We repeat this procedure every time we sample an image from the replay buffer. The plots show that image augmentation significantly reduces overfitting, closing the performance gap between the encoder architectures. These random shifts alone enable SAC to achieve competitive performance without the need for auxiliary losses.

5.3.2 Optimality Invariant Image Transformations

While the image augmentation described above is effective, it does not fully exploit the MDP structure inherent in RL tasks. We now introduce a general framework for regularizing the value function through transformations of the input state. For a given task, we define an optimality invariant state transformation $f : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ as a mapping that preserves the Q-values

$$Q(s, a) = Q(f(s, \nu), a) \text{ for all } s \in \mathcal{S}, a \in \mathcal{A} \text{ and } \nu \in \mathcal{T}.$$

where ν are the parameters of $f(\cdot)$, drawn from the set of all possible parameters $\mathcal{T}$. One example of such transformations is the random image translations successfully applied in the previous section.

For every state, the transformations allow the generation of several surrogate states with the same Q-values, thus providing a mechanism to reduce the variance of Q-function estimation. In particular, for an arbitrary distribution of states $\mu(\cdot)$ and policy π, instead

of using a single sample $s^* \sim \mu(\cdot)$, $a^* \sim \pi(\cdot|s^*)$ estimation of the following expectation

$$\mathop{\mathbb{E}}_{\substack{s\sim\mu(\cdot)\\ a\sim\pi(\cdot|s)}}[Q(s,a)] \approx Q(s^*,a^*)$$

we can instead generate K samples via random transformations and obtain an estimate with lower variance

$$\mathop{\mathbb{E}}_{\substack{s\sim\mu(\cdot)\\ a\sim\pi(\cdot|s)}}[Q(s,a)] \approx \frac{1}{K}\sum_{k=1}^{K} Q(f(s^*,\nu_k),a_k) \text{ where } \nu_k \in \mathcal{T} \text{ and } a_k \sim \pi(\cdot|f(s^*,\nu_k)).$$

This suggests two distinct ways to regularize Q-function. First, we use the data augmentation to compute the target values for every transition tuple (s_i, a_i, r_i, s'_i) as

$$y_i = r_i + \gamma\frac{1}{K}\sum_{k=1}^{K} Q_\theta(f(s'_i,\nu'_{i,k}),a'_{i,k}) \text{ where } a'_{i,k} \sim \pi(\cdot|f(s'_i,\nu'_{i,k})) \tag{5.1}$$

where $\nu'_{i,k} \in \mathcal{T}$ corresponds to a transformation parameter of s'_i. Then the Q-function is updated using these targets through an SGD update using learning rate λ_θ

$$\theta \leftarrow \theta - \lambda_\theta \nabla_\theta \frac{1}{N}\sum_{i=1}^{N}(Q_\theta(f(s_i,\nu_i),a_i) - y_i)^2. \tag{5.2}$$

In tandem, we note that the same target from eq. (5.1) can be used for different augmentations of s_i, resulting in the second regularization approach

$$\theta \leftarrow \theta - \lambda_\theta \nabla_\theta \frac{1}{NM}\sum_{i=1,m=1}^{N,M}(Q_\theta(f(s_i,\nu_{i,m}),a_i) - y_i)^2. \tag{5.3}$$

When both regularization methods are used, $\nu_{i,m}$ and $\nu'_{i,k}$ are drawn independently.

5.3.3 Our approach: Data-regularized Q (DrQ)

Our approach, **DrQ**, is the union of the three separate regularization mechanisms introduced above:

1. transformations of the input image (section 5.3.1).
2. averaging the Q target over K image transformations (eq. (5.1)).
3. averaging the Q function itself over M image transformations (eq. (5.3)).

algorithm 5.1 details how they are incorporated into a generic pixel-based off-policy actor-critic algorithm. If [K=1,M=1] then **DrQ** reverts to *image transformations alone*, this makes applying **DrQ** to any model-free RL algorithm straightforward as it does not require any modifications to the algorithm itself. Note that **DrQ** [K=1,M=1] also exactly recovers the concurrent work of RAD (Laskin et al., 2020), up to a particular choice of hyper-parameters and data augmentation type.

For the experiments in this paper, we pair **DrQ** with SAC (Haarnoja et al., 2018b) and DQN (Mnih et al., 2013), popular model-free algorithms for control in continuous and discrete action spaces respectively. We select image shifts as the class of image transformations f, with $\nu \pm 4$, as explained in section 5.3.1. For target Q and Q augmentation we use [K=2,M=2] respectively. fig. 5.4 shows **DrQ** and ablated versions, demonstrating clear gains over unaugmented SAC.

5.4 Experiments

In this section we evaluate our algorithm (**DrQ**) on the two commonly used benchmarks based on the DeepMind control suite (Tassa et al., 2018): the PlaNet (Hafner et al., 2018) and Dreamer (Hafner et al., 2019) setups. We keep all hyper-parameters

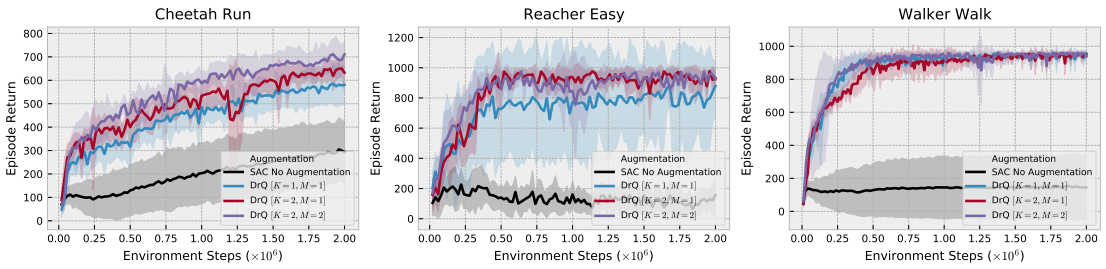


Figure 5.4: Different combinations of our three regularization techniques on tasks from (Tassa et al., 2018) using SAC. Black: standard SAC. Blue: **DrQ** [K=1,M=1], SAC augmented with random shifts. Red: **DrQ** [K=2,M=1], random shifts + Target Q augmentations. Purple: **DrQ** [K=2,M=2], random shifts + Target Q + Q augmentations. All three regularization methods correspond to Algorithm 1 with different hyperparameters K,M and independently provide beneficial gains over unaugmented SAC. Note that **DrQ** [K=1,M=1] exactly recovers the concurrent work of RAD (Laskin et al., 2020) up to a particular choice of hyper-parameters and data augmentation type.

throughout these experiments fixed: we train the actor and critic neural networks using the Adam optimizer (Kingma and Ba, 2014) with default parameters and a mini-batch size of 512. For SAC, the soft target update rate τ is 0.01, initial temperature is 0.1, and target network and we perform one actor update every 2 critic updates (as in Fujimoto et al. (2018b)). We use the image encoder architecture from SAC-AE (Yarats et al., 2019) and follow their training procedure.

Following Henderson et al. (2018), we train the models using 10 different seeds; for every seed, we compute the mean episode returns every 10000 environment steps, averaging over 10 episodes. All figures plot the mean performance over the 10 seeds, together with ± 1 standard deviation shading. We compare our **DrQ** approach to leading model-free and model-based approaches: PlaNet (Hafner et al., 2018), SAC-AE (Yarats et al., 2019), SLAC (Lee et al., 2019), CURL (Srinivas et al., 2020) and Dreamer (Hafner et al., 2019). The comparisons use the results provided by the authors of the corresponding papers.

Algorithm 5.1 DrQ: **D**ata-**r**egularized **Q** applied to a generic off-policy actor critic algorithm.
Black: unmodified off-policy actor-critic.
Orange: image transformation.
Green: target Q augmentation.
Blue: Q augmentation.

Hyperparameters: Total number of environment steps T, mini-batch size N, learning rate λ_θ, target network update rate τ, image transformation f, number of target Q augmentations K, number of Q augmentations M.
for each timestep $t = 1..T$ **do**
  $a_t \sim \pi(\cdot|s_t)$
  $s'_t \sim p(\cdot|s_t, a_t)$
  $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, r(s_t, a_t), s'_t)$
  UPDATECRITIC($\mathcal{D}$)
  UPDATEACTOR($\mathcal{D}$) ▷ Data augmentation is applied to the samples for actor training as well.
procedure UPDATECRITIC($\mathcal{D}$)
  $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N \sim \mathcal{D}$ ▷ Sample a mini batch
  $\{\nu'_{i,k} | \nu'_{i,k} \sim \mathcal{U}(\mathcal{T}), i = 1..N, k = 1..K\}$ ▷ Target augmentations
  for each $i = 1..N$ **do**
    $a'_i \sim \pi(\cdot|s'_i)$ or $a'_{i,k} \sim \pi(\cdot|\, f(s'_i, \nu'_{i,k})), k = 1..K$
    $\hat{Q}_i = Q_{\theta'}(s'_i, a'_i)$ or $\hat{Q}_i = \frac{1}{K}\sum_{k=1}^K Q_{\theta'}(f(s'_i, \nu'_{i,k}), a'_{i,k})$
    $y_i \leftarrow r(s_i, a_i) + \gamma \hat{Q}_i$
  $\{\nu_{i,m} | \nu_{i,m} \sim \mathcal{U}(\mathcal{T}), i = 1..N, m = 1..M\}$ ▷ Q augmentations
  $J_Q(\theta) = \frac{1}{N}\sum_{i=1}^N (Q_\theta(s_i, a_i) - y_i)^2$
  or $J_Q(\theta) = \frac{1}{NM}\sum_{i,m=1}^{N,M} (Q_\theta(f(s_i, \nu_{i,m}), a_i) - y_i)^2$
  $\theta \leftarrow \theta - \lambda_\theta \nabla_\theta J_Q(\theta)$ ▷ Update the critic
  $\theta' \leftarrow (1 - \tau)\theta' + \tau\theta$ ▷ Update the critic target

5.4.1 DeepMind Control Suite Experiments

We consider two evaluation setups that were introduced in PlaNet (Hafner et al., 2018) and Dreamer (Hafner et al., 2019), both using tasks from the DeepMind control suite (Tassa et al., 2018). The PlaNet benchmark consists of six tasks of various traits. Importantly, the benchmark proposed to use a different action repeat hyper-parameter for each task, which we summarize in table 5.2.

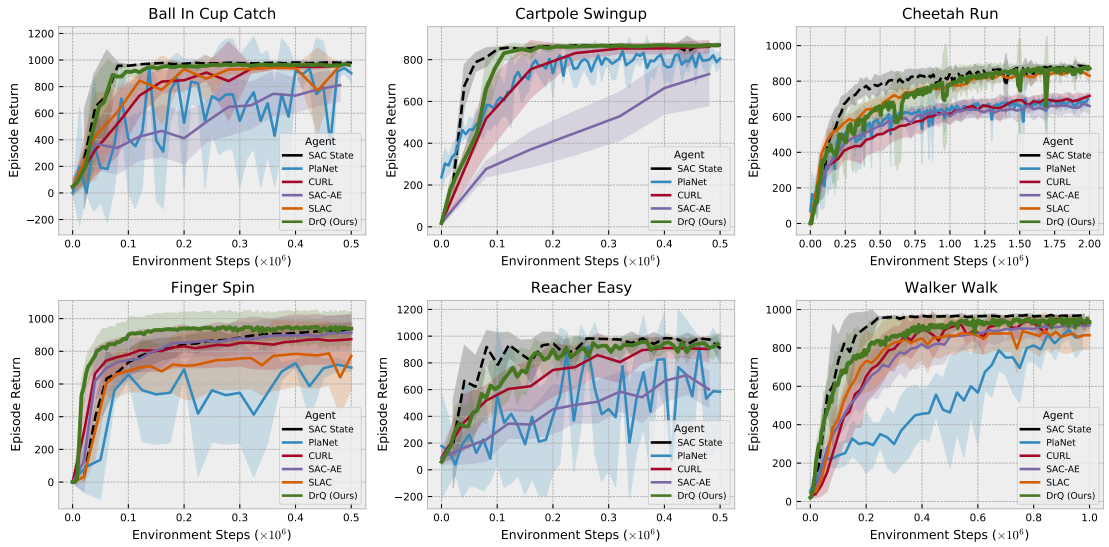


Figure 5.5: The PlaNet benchmark. Our algorithm (**DrQ** [K=2,M=2]) outperforms the other methods and demonstrates the state-of-the-art performance. Furthermore, on several tasks **DrQ** is able to match the upper-bound performance of SAC trained directly on internal state, rather than images. Finally, our algorithm not only shows improved sample-efficiency relative to other approaches, but is also faster in terms of wall clock time.

The Dreamer benchmark considers an extended set of tasks, which makes it more difficult that the PlaNet setup. Additionally, this benchmark requires to use the same set hyper-parameters for each task, including action repeat (set to 2), which further increases the difficulty.

We employ an encoder architecture from Yarats et al. (2019) inspired by the D4PG architecture from Tassa et al. (2018). This encoder consists of four convolutional layers with 3×3 kernels and 32 channels. The `ReLU` activation is applied after each conv layer. We use stride to 1 everywhere, except of the first conv layer, which has stride 2. The output of the convnet is feed into a single fully-connected layer normalized by `LayerNorm` (Ba et al., 2016). Finally, we apply `tanh` nonlinearity to the 50 dimensional output of the fully-connected layer. We initialize the weight matrix of fully-connected and convolutional layers with the orthogonal initialization (Saxe et al.,

500k step scores	DrQ (Ours)	CURL	PlaNet	SLAC	SAC State
Finger Spin	**938±103**	874±151	718±40	771±203	927±43
Cartpole Swingup	**868±10**	861±30	787±46	-	870±7
Reacher Easy	**942±71**	904±94	588±471	-	975±5
Cheetah Run	**660±96**	500±91	568±21	629±74	772±60
Walker Walk	**921±45**	906±56	478±164	865±97	964±8
Ball In Cup Catch	**963±9**	958±13	939±43	959±4	979±6
100k step scores					
Finger Spin	**901±104**	779±108	560±77	680±130	672±76
Cartpole Swingup	**759±92**	592±170	563±73	-	812±45
Reacher Easy	**601±213**	517±113	82±174	-	919±123
Cheetah Run	344±67	307±48	165±123	**391±47***	228±95
Walker Walk	**612±164**	344±132	221±43	428±74	604±317
Ball In Cup Catch	**913±53**	772±241	710±217	607±173	957±26

Table 5.1: The PlaNet benchmark at 100k and 500k environment steps. Our method (**DrQ** [K=2,M=2]) outperforms other approaches in both the data-efficient (100k) and asymptotic performance (500k) regimes. *: SLAC uses 100k exploration steps which are not counted in the reported values. By contrast, **DrQ** only uses 1000 exploration steps which are included in the overall step count.

Table 5.2: The action repeat hyper-parameter used for each task in the PlaNet benchmark.

Task name	Action repeat
Cartpole Swingup	8
Reacher Easy	4
Cheetah Run	4
Finger Spin	2
Ball In Cup Catch	4
Walker Walk	2

2013) and set the bias to be zero.

The actor and critic networks both have separate encoders, although we share the weights of the conv layers between them. Furthermore, only the critic optimizer is allowed to update these weights (e.g. we stop the gradients from the actor before they propagate to the shared conv layers). We summarize all hyperparameters used for the experiments on DeepMind control suite in **??**.

Our agent first collects 1000 seed observations using a random policy. The further training observations are collected by sampling actions from the current policy. We perform one training update every time we receive a new observation. In cases where we use action repeat, the number of training observations is only a fraction of the environment steps (e.g. a 1000 steps episode at action repeat 4 will only results into 250 training observations). We evaluate our agent every 10000 true environment steps by computing the average episode return over 10 evaluation episodes. During evaluation we take the mean policy action instead of sampling.

PlaNet Benchmark (Hafner et al., 2018) consists of six challenging control tasks from (Tassa et al., 2018) with different traits. The benchmark specifies a different action-repeat hyper-parameter for each of the six tasks[1]. Following common practice (Hafner et al., 2018; Lee et al., 2019; Mnih et al., 2013; Yarats et al., 2019), we report the performance using true environment steps, thus are invariant to the action-repeat hyper-parameter. Aside from action-repeat, our algorithm's hyper-parameters are fixed across the six tasks, using the values previously detailed (see **??**).

Figure 5.5 compares **DrQ** [K=2,M=2] to PlaNet (Hafner et al., 2018), SAC-AE (Yarats et al., 2019), CURL (Srinivas et al., 2020), SLAC (Lee et al., 2019), and an upper bound performance provided by SAC (Haarnoja et al., 2018b) that directly learns from internal states. We use the version of SLAC that performs one gradient update per an environment step to ensure a fair comparison to other approaches. **DrQ** achieves state-of-the-art performance on this benchmark on all the tasks, despite being much simpler than other methods. Furthermore, since **DrQ** does not learn a model (Hafner et al., 2018; Lee et al., 2019) or any auxiliary tasks (Srinivas et al., 2020), the wall clock time also compares favorably to the other methods. In table 5.1 we also compare performance given at a

[1]This means the number of training observations is a fraction of the environment steps (e.g. an episode of 1000 steps with action-repeat 4 results in 250 training observations).

fixed number of environment interactions (e.g. 100k and 500k).

Dreamer Benchmark (Hafner et al., 2019) is a more extensive testbed, featuring a diverse set of tasks from the DeepMind control suite. We exclude tasks involving sparse reward (e.g., Acrobot and Quadruped) since they require SAC modification to incorporate multi-step returns (Barth-Maron et al., 2018), which is beyond the scope of this work. We evaluate **DrQ** on the remaining 15 tasks, fixing the action-repeat hyper-parameter to 2, as in Dreamer (Hafner et al., 2019).

We compare **DrQ** [K=2,M=2] to Dreamer (Hafner et al., 2019) and the upper-bound performance of SAC (Haarnoja et al., 2018b) from states[2]. Again, we keep all the hyper-parameters of our algorithm fixed across all the tasks. In fig. 5.6, **DrQ** demonstrates the state-of-the-art results by collectively outperforming Dreamer (Hafner et al., 2019), although Dreamer is superior on 3 of the 15 tasks (Walker Run, Cartpole Swingup Sparse, and Pendulum Swingup). On many tasks, **DrQ** approaches the upper-bound performance of SAC (Haarnoja et al., 2018b) trained directly on states.

5.4.2 Atari 100k Experiments

We evaluate **DrQ** [K=1,M=1] on the recently introduced Atari 100k (Kaiser et al., 2019) benchmark – a sample-constrained evaluation for discrete control algorithms. The underlying RL approach to which **DrQ** is applied is a DQN, combined with double Q-learning (van Hasselt et al., 2015), n-step returns (Mnih et al., 2016), and dueling critic architecture (Wang et al., 2015). We largely reuse the hyper-parameters from OTRainbow (Kielak, 2020), but adapt them for DQN (Mnih et al., 2013).

As per common practice (Kaiser et al., 2019; van Hasselt et al., 2019), we evaluate

[2]No other publicly reported results are available for the other methods due to the recency of the Dreamer (Hafner et al., 2019) benchmark.

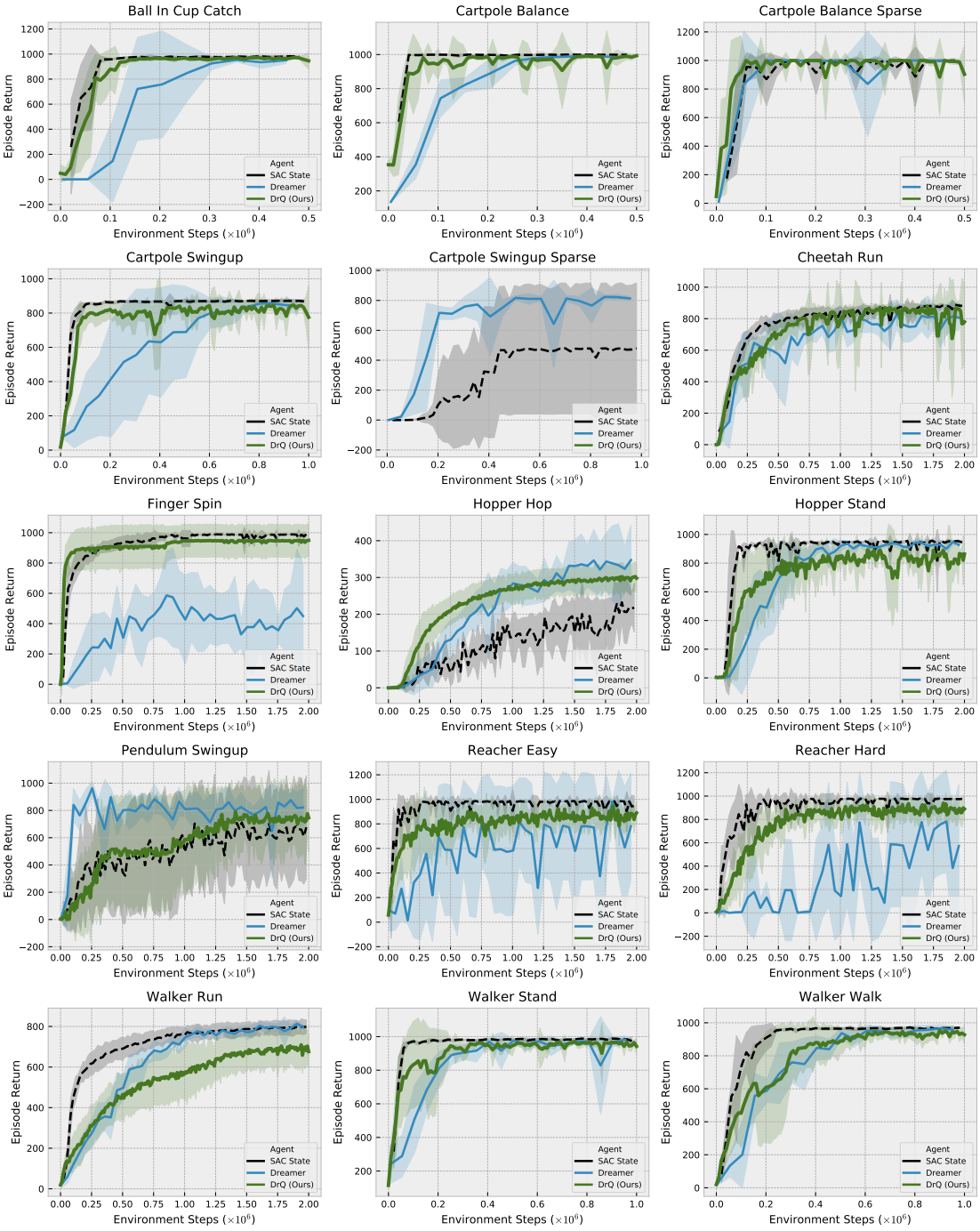


Figure 5.6: The Dreamer benchmark. Our method (**DrQ** [K=2,M=2]) again demonstrates superior performance over Dreamer on 12 out 15 selected tasks. In many cases it also reaches the upper-bound performance of SAC that learns directly from states.

our agent for 125k environment steps at the end of training and average its performance over 5 random seeds. fig. 5.7 shows the median human-normalized episode returns performance (as in (Mnih et al., 2013)) of the underlying model, which we refer to as Efficient DQN, in pink. When we add **DrQ** , there is a significant increase in performance (cyan), surpassing OTRainbow (Kielak, 2020) and Data Efficient Rainbow (van Hasselt et al., 2019). **DrQ** is also superior to CURL (Srinivas et al., 2020) that uses an auxiliary loss built on top of a hybrid between OTRainbow and Efficient rainbow. **DrQ** combined with Efficient DQN achieves state-of-the-art performance, despite being significantly simpler than the other approaches.

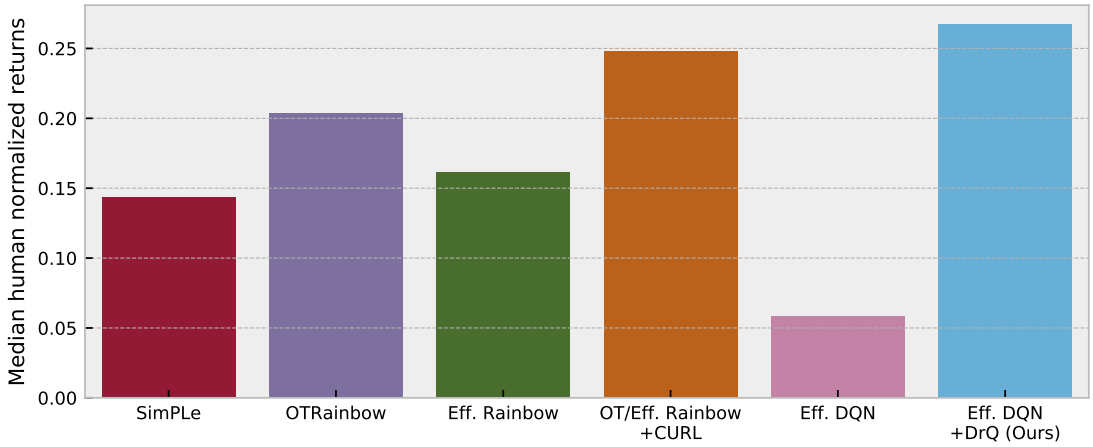


Figure 5.7: The Atari 100k benchmark. Compared to a set of leading baselines, our method (**DrQ** [K=1,M=1], combined with Efficient DQN) achieves the state-of-the-art performance, despite being considerably simpler. Note the large improvement that results from adding **DrQ** to Efficient DQN (pink vs cyan). By contrast, the gains from CURL, that utilizes tricks from both Data Efficient Rainbow and OTRainbow, are more modest over the underlying RL methods.

Besides reporting in fig. 5.7 median human-normalized episode returns over the 26 Atari games used in (Kaiser et al., 2019), we also provide the mean episode return for each individual game in table 5.3.

Table 5.3: Mean episode returns on each of 26 Atari games from the setup in Kaiser et al. (2019). The results are recorded at the end of training and averaged across 5 random seeds (the CURL's results are averaged over 3 seeds as reported in Srinivas et al. (2020)). On each game we mark as bold the highest score. Our method demonstrates better overall performance (as reported in fig. 5.7).

Game	Rainbow	SimPLe	OTRainbow	Eff. Rainbow	Eff. Rainbow +CURL	Eff. DQN	Eff. DQN +DrQ (Ours)
Alien	318.7	616.9	824.7	739.9	**1148.2**	558.1	702.5
Amidar	32.5	88.0	82.8	188.6	**232.3**	63.7	100.2
Assault	231.0	527.2	351.9	431.2	543.7	**589.5**	490.3
Asterix	243.6	**1128.3**	628.5	470.8	524.3	341.9	577.9
BankHeist	15.6	34.2	182.1	51.0	193.7	74.0	**205.3**
BattleZone	2360.0	5184.4	4060.6	10124.6	**11208.0**	4760.8	6240.0
Boxing	-24.8	**9.1**	2.5	0.2	4.8	-1.8	5.1
Breakout	1.2	16.4	9.8	1.9	**18.2**	7.3	14.3
ChopperCommand	120.0	**1246.9**	1033.3	861.8	1198.0	624.4	870.1
CrazyClimber	2254.5	**62583.6**	21327.8	16185.3	27805.6	5430.6	20072.2
DemonAttack	163.6	208.1	711.8	508.0	834.0	403.5	**1086.0**
Freeway	0.0	20.3	25.0	**27.9**	**27.9**	3.7	20.0
Frostbite	60.2	254.7	231.6	866.8	**924.0**	202.9	889.9
Gopher	431.2	771.0	778.0	349.5	**801.4**	320.8	678.0
Hero	487.0	2656.6	6458.8	**6857.0**	6235.1	2200.1	4083.7
Jamesbond	47.4	125.3	112.3	301.6	**400.1**	133.2	330.3
Kangaroo	0.0	323.1	605.4	779.3	345.3	448.6	**1282.6**
Krull	1468.0	**4539.9**	3277.9	2851.5	3833.6	2999.0	4163.0
KungFuMaster	0.0	**17257.2**	5722.2	14346.1	14280.0	2020.9	7649.0
MsPacman	67.0	1480.0	941.9	1204.1	**1492.8**	872.0	1015.9
Pong	-20.6	**12.8**	1.3	-19.3	2.1	-19.4	-17.1
PrivateEye	0.0	58.3	100.0	97.8	105.2	**351.3**	-50.4
Qbert	123.5	**1288.8**	509.3	1152.9	1225.6	627.5	769.1
RoadRunner	1588.5	5640.6	2696.7	**9600.0**	6786.7	1491.9	8296.3
Seaquest	131.7	**683.3**	286.9	354.1	408.0	240.1	299.4
UpNDown	504.6	**3350.3**	2847.6	2877.4	2735.2	2901.7	3134.8
Median episode returns (human normalized)	0.020	0.135	0.208	0.147	0.240	0.094	**0.270**

5.5 Related Work

Computer Vision Data augmentation via image transformations has been used to improve generalization since the inception of convolutional networks (Becker and Hinton, 1992; Ciregan et al., 2012; Ciresan et al., 2011; LeCun et al., 1989). Following AlexNet (Krizhevsky et al., 2012), they have become a standard part of training pipelines.

For object classification tasks, they select the transformations to avoid changing the semantic category, i.e., translations, scales, and color shifts. They use perturbed versions of input examples to expand the training set, and no adjustment to the training algorithm is needed. While a similar set of transformations are potentially applicable to control tasks, the RL context does not require us to modify the underlying algorithm.

Data augmentation methods have also been used in the context of self-supervised learning. (Dosovitskiy et al., 2016) use per-exemplar perturbations in an unsupervised classification framework. More recently, several approaches (Chen et al., 2020; He et al., 2019; Hénaff et al., 2019; Misra and van der Maaten, 2019) have used invariance to imposed image transformations in contrastive learning schemes, producing state-of-the-art results on downstream recognition tasks. By contrast, our scheme addresses control tasks, utilizing different types of invariance.

Regularization in RL Some early attempts to learn RL function approximators used ℓ_2 regularization of the Q (Farahmand et al., 2008; Yan et al., 2017) function. Another approach is entropy regularization (Haarnoja et al., 2018b; Nachum et al., 2017; Williams and Peng, 1991; Ziebart et al., 2008), where causal entropy is added to the rewards, making the Q-function smoother and facilitating optimization (Ahmed et al., 2018). Prior work has explored the neural network approximator's regularization in deep RL, e.g., using dropout (Farebrother et al., 2018) and cutout (Cobbe et al., 2018) techniques. See (Liu et al., 2019) for a comprehensive evaluation of different network regularization methods. In contrast, our approach directly regularizes the Q-function in a data-driven way that incorporates knowledge of task invariances instead of generic priors.

Generalization between Tasks and Domains A range of recently introduced datasets explicit aim to improving generalization in RL through deliberate variation of the scene colors/textures/backgrounds/viewpoints. These include robot learning in homes (Gupta

et al., 2018), simulated robotics tasks (Yu et al., 2019), procedural generated games (Cobbe et al., 2019). There are also domain randomization techniques (Slaoui et al., 2019; Tobin et al., 2017) which synthetically apply similar variations but assume control of the data generation procedure, in contrast to our method. Furthermore, these works address generalization between domains (e.g., synthetic-to-real or different game levels), whereas our work focuses on a single domain and task. In concurrent work, RAD (Laskin et al., 2020) also demonstrates that image augmentation can improve sample efficiency and generalization of RL algorithms. However, RAD represents our algorithm's specific instantiation when [K=1,M=1] and uses different image augmentations.

Continuous Control from Pixels Various methods address the sample-efficiency of RL algorithms that directly learn from pixels. The most prominent approaches for this fall into two categories, model-based and model-free methods. The model-based methods attempt to learn the system dynamics to acquire a compact latent representation of high-dimensional observations to perform policy search (Hafner et al., 2019, 2018; Lee et al., 2019). In contrast, the model-free methods either learn the latent representation indirectly by optimizing the RL objective (Abdolmaleki et al., 2018; Barth-Maron et al., 2018) or by employing auxiliary losses that provide additional supervision (Dwibedi et al., 2018; Sermanet et al., 2018; Srinivas et al., 2020; Yarats et al., 2019). Our approach is complementary to these methods and can be combined with them to improve performance.

5.6 Conclusion

We have introduced a simple regularization technique that significantly improves SAC's performance trained directly from image pixels on standard continuous control

tasks. Our method is easy to implement and adds a little computational burden. We compared our method to state-of-the-art approaches on both DeepMind control suite, where we demonstrated that it outperforms them on the majority of tasks, and Atari 100k benchmarks, where it outperforms other methods in the median metric. Furthermore, we demonstrate the method to be robust to the choice of hyper-parameters.

Chapter 6

Offline Reinforcement Learning with Fisher Divergence Critic Regularization

Many modern approaches to offline Reinforcement Learning (RL) utilize *behavior regularization*, typically augmenting a model-free actor-critic algorithm with a penalty measuring divergence of the policy from the offline data. In this work, we propose an alternative approach to encouraging the learned policy to stay close to the data, namely parameterizing the critic as the $\log$-behavior-policy, which generated the offline data, plus a state-action value offset term, which we learn using a neural network. Behavior regularization then corresponds to an appropriate regularizer on the offset term. We propose using a gradient penalty regularizer for the offset term and demonstrate its equivalence to Fisher divergence regularization, suggesting connections to the score matching and generative energy-based model literature. We thus term our resulting algorithm Fisher-BRC (Behavior Regularized Critic). On standard offline RL benchmarks,

Fisher-BRC achieves both improved performance and faster convergence over existing state-of-the-art methods.

6.1 Introduction

Reinforcement learning (RL) describes the field of machine learning concerned with learning a policy to solve a task through stochastic trial-and-error experience in an environment. The typically assumed default setting in RL is of *online* access to the environment; i.e., the learned policy may collect new trial-and-error experience directly from the environment. However, in many practical scenarios, where deploying a new policy to interact with the live environment is expensive or associated with risks or safety concerns (Thomas, 2015), it is more common to have only *offline* access to the environment. That is, the trial-and-error experience available for learning a task-solving policy is a static, offline dataset of experience collected by some other *behavior* policy. This setting is known as offline RL and has attracted a significant amount of interest in recent years (Lange et al., 2012; Levine et al., 2020).

Many of the recent approaches to offline RL utilize some form of *behavior regularization*, which compels the learned policy to stay close to the data-generating behavior policy (Wu et al., 2019). For example, in model-free actor-critic algorithms, behavior regularization is typically done by augmenting the actor loss with a penalty measuring the learned policy's divergence from the behavior policy (Jaques et al., 2019; Kumar et al., 2019; Wu et al., 2019), reminiscent of KL-control methods in related literature (Jaques et al., 2017; Kappen et al., 2012). While straightforward, a disadvantage of this approach is that it does little to regularize the critic itself. Thus, it is common for the critic to take on wildly extrapolated values on actions unseen in the training data, which may dominate

any behavior regularization applied to the actor, as we demonstrate in this work.

In this work, we propose an alternative approach to encouraging the learned policy to stay close to the offline data. Focusing on the critic, we propose parameterizing the critic values as the behavior policy's logits plus an additional offset term. When we train the actor (the learned policy) to choose actions that maximize the critic value, it will be compelled to stay close to the behavior policy as long as the offset term is suitably 'small'. Thus, behavior regularization corresponds to augmenting the standard Bellman error critic loss with an appropriate regularization on the offset term in this setting.

What should this regularization term be? After noting that, in continuous control, the offset term's effect on the learned policy is via the gradients of the offset with respect to actions, we propose regularizing the offset term with a gradient penalty. While this may appear heuristic at first, we present mathematical derivations establishing a connection between this gradient penalty and the *Fisher divergence*, which appears in the score matching and energy-based generative model literature (Bao et al., 2020; Lyu, 2012), interpreting the critic values as the energy function of a Boltzmann distribution.

We thus term our newly proposed actor-critic algorithm *Fisher-BRC* (behavior regularized critic). To aid the conceptual understanding of Fisher-BRC, we analyze its training dynamics in a simple toy setting, highlighting the advantage of its implicit Fisher divergence regularization instead of the more explicit divergence penalties imposed by alternative offline RL methods. We then present an extensive evaluation of Fisher-BRC on standard offline RL benchmarks. We find that Fisher-BRC yields state-of-the-art performance compared to a variety of existing model-free and model-based RL methods. We further show that while Fisher-BRC learns better policies than other methods, it is also much more computationally efficient than more sophisticated offline RL algorithms. Overall, Fisher-BRC presents a new approach to behavior regularization in offline RL

with compelling practical benefits.

6.2 Related Work

Our work adds to the rich literature on behavior regularization methods in offline RL (Wu et al., 2019), which propose several regularizations in RL training that compel the learned policy to stay close to the offline data. These regularizers have appeared as divergence penalties (Jaques et al., 2019; Kumar et al., 2019; Wu et al., 2019), implicitly through appropriate network initializations (Matsushima et al., 2020), or more explicitly through careful parameterization of the policy (Fujimoto et al., 2019). Another way to apply behavior regularizers is via modification of the critic learning objective as in Kumar et al. (2020); Nachum et al. (2019c). Our work is unique in applying behavior regularization through a parameterization of the critic, and empirically, we find this to yield much better performance.

This work demonstrates that the specific parameterization we employ has connections to Fisher divergence regularization, establishing connections with energy-based models. Prior methods have proposed using energy-based models for policies to express more multi-modal distributions (Haarnoja et al., 2017; Heess et al., 2013). Meanwhile, while our work focuses on reinforcement learning – i.e., learning a return-maximizing policy – other works have established connections between energy-based models and inverse RL or imitation learning (Finn et al., 2016a).

In practice, our regularization reduces to a gradient penalty applied to the offset term in the critic. Gradient penalties have appeared in previous work, arguably first popularized in machine learning by Wasserstein generative-adversarial networks (Arjovsky et al., 2017), but also used in imitation learning (chapter 3 and chapter 4), cross-domain disen-

tanglement (Gonzalez-Garcia et al., 2018), and uncertainty estimation (van Amersfoort et al., 2020).

6.3 Background

We introduce the notation and assumptions used in this paper and provide an in-depth review of the methods most closely related to ours.

6.3.1 Reinforcement Learning

In this work, we consider environments that can be represented as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, p_0, p, r, \gamma)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $p_0(s)$ is a distribution of initial states, $p(s'|s, a)$ is a stochastic dynamics model, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function and $\gamma \in [0, 1)$ is a discount. We restrict our work to continuous action spaces; i.e., $\mathcal{A} \subset \mathbb{R}^d$ for some d. The goal of reinforcement learning is to find a policy $\pi(a|s)$ that maximizes the cumulative discounted returns $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 \sim p_0(\cdot), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)]$. In online reinforcement learning, it is usually assumed that an agent learns based on experience generated by the agent interacting with the learning environment.

6.3.2 Offline Reinforcement Learning

In this work, we focus on the offline setting, in which the agent cannot generate new experience data and learns based on a provided dataset $\mathcal{D}$ of (s, a, r, s') tuples generated by some other policy interacting with the environment. We call the policy that generated this dataset the *behavior* policy and denotes it as μ.

Offline datasets usually do not provide complete state-action coverage. That is, the set $\{(s, a) \mid (s, a, r, s') \in \mathcal{D}\}$ is typically a small subset of the full space $\mathcal{S} \times \mathcal{A}$. Standard reinforcement learning methods such as SAC (Haarnoja et al., 2018b) or DDPG (Lillicrap et al., 2015) suffer when applied to these datasets due to extrapolation errors (Fujimoto et al., 2019; Kumar et al., 2019).

Behavior regularization is a prominent approach to offline reinforcement learning that aims to address this problem by using appropriate regularizers to compel the learned policy to stay close to the data. There are two common ways to incorporate behavior regularization into the actor-critic framework – via policy regularization or a critic penalty. Since our approach is related to both techniques, in the following sections, we describe the two approaches and problems associated with them.

6.3.3 Policy Regularization

Policy regularization can be imposed either during critic or policy learning. First, we describe a family of approaches based on applying behavior constraints to training policies. These constraints can be applied in a hard fashion, by restricting the policy action space to the actions seen in the offline dataset as in BCQ Fujimoto et al. (2019):

$$\pi(s) := \underset{a_i + \xi_\phi(s, a_i, \Phi)}{\arg\max} \; Q_\theta(s, a_i + \xi_\phi(s, a_i, \Phi)), \{a_i \sim \mu(\cdot|s)\}_{i=1}^{n}$$

where $\pi(s)$ is a deterministic policy, $\xi_\phi(s, a, \Phi)$ is a perturbation model on actions constrained to the interval $[-\Phi, \Phi]$, $\mu(a|s)$ is a behavioral policy contracted by fitting a density model to the offline dataset. The hyperparameter n controls the number of sampled actions. In other words, one samples n perturbed actions from the behavior policy and chooses from the action with the largest critic-approximated value. One of the

limitations of this approach is that a large number of sampled actions might be required for competitive performance (Ghasemipour et al., 2021).

Using a divergence penalty is an alternative approach to policy regularization. Instead of having hard constraints on the training policy, one can regularize the policy with an appropriately chosen probability divergence such as the KL-divergence (Jaques et al., 2019; Wu et al., 2019):

$$\max_{\pi} \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)} [Q_\theta(s,a)] - \alpha D_{KL}(\pi(\cdot|s)\|\mu(\cdot|s)) \right]. \tag{6.1}$$

Although these approaches demonstrate impressive performance on some tasks, they share a common problem. The Q-function, Q_θ, learned via standard TD-error minimization on $\mathcal{D}$ receives no learning signal for actions not observed in the replay buffer, while this same Q-function is nevertheless queried on out-of-distribution actions during the policy update – i.e., when $Q_\theta(s,a)$ is evaluated on $a \sim \pi(\cdot|s)$ in eq. (6.1) – and for bootstrapping critic targets in the squared TD-loss:

$$J(Q_\theta) := \mathop{\mathbb{E}}_{\substack{(s,a,s')\sim\mathcal{D} \\ a'\sim\pi(\cdot|s')}} [(r(s,a) + \gamma Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2]. \tag{6.2}$$

Thus, issues with critic extrapolation can still dominate divergence regularizers applied to the policy.

6.3.4 Critic Penalty

Other works, such as AlgaeDICE Nachum et al. (2019c) and CQL Kumar et al. (2020) attempt to incorporate some divergence regularization into the critic. In particular, AlgaeDICE introduces a term that pushes Q-values down for actions sampled from the

training policy while minimizing TD-error via residual learning:

$$\min_{\theta} \alpha(1-\gamma) \mathop{\mathbb{E}}_{\substack{s_0 \sim \pi_0(\cdot) \\ a_0 \sim \pi(s_0)}} [Q_\theta(s_0, a_0)] +$$

$$\mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}} [(r(s,a) + \gamma \mathop{\mathbb{E}}_{\substack{s' \sim p(\cdot|s,a) \\ a' \sim \pi(\cdot|s')}} [Q_\theta(s', a')] - Q_\theta(s,a))^2].$$

This formulation can be generalized by replacing the squared function with some convex function f. The choice of f corresponds to an implicit f-divergence regularization on the learned policy with respect to state-action distributions. For example, a choice of $f(x) = \exp(x-1)$ or $f(x) = \log \mathbb{E}_{\mathcal{D}} \exp(x)$ corresponds to an implicit KL-divergence.

Based on a similar idea, CQL (Kumar et al., 2020) extends the standard critic loss $J(Q_\theta)$ in eq. (6.2) with additional terms that minimize Q-values sampled from a policy and maximize values of the dataset actions:

$$\min_{\theta} J(Q_\theta) + \lambda \mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}} [\log \Sigma_a \exp(Q_\theta(s,a)) - Q_\theta(s,a)]. \tag{6.3}$$

Although both of these methods provide a learning signal to the critic Q-values on the entire action space, AlgaeDICE is based on residual learning with slower convergence than fitted TD-learning (Baird, 1995). On the other hand, although CQL can achieve better empirical performance, the log-sum-exp term that appears in its formulation is not tractable for continuous actions and must be computed via numerical integration. The authors of CQL propose doing this via Monte-Carlo sampling with importance weights, where they use the current training policy to draw samples for the procedure. This process can add a significant computational burden to actor-critic training. In contrast, we will show that our proposed Fisher-BRC achieves good empirical performance while maintaining computational efficiency closer to standard actor critics.

6.4 Fisher-BRC

We now continue to describe our approach to behavior regularization in offline RL settings, which aims to circumvent the issues associated with competing methods described in the previous section, namely (1) lack of well-defined critic values on out-of-distribution actions and (2) computational inefficiency of critic penalty approaches. We begin with a conceptual derivation of our method before presenting a more formal connection to Fisher divergence regularization. See algorithm 6.1 for a sketch of the algorithm.

Algorithm 6.1 Fisher-BRC [Sketch].

Input: Dataset $\mathcal{D}$, offset network O_θ, policy network π_ϕ.

1. Learn approximate μ using behavioral cloning.
2. Update θ using objective in eq. (6.7).
3. Update ϕ using entropy-regularized objective $\mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\phi(\cdot|s)}[O_\theta(s,a) + \log\mu(a|s) + \alpha\mathcal{H}(\pi_\phi(\cdot|s))]$.
4. Repeat from 2.

Return: π_ϕ.

6.4.1 Conceptual Derivation

We start from the observation that we can represent the entropy smoothed Q-values of the behavior policy $\mu(\cdot|s)$ as

$$Q(s,a) = V(s) + \log\mu(a|s).$$

This decomposition of Q-values into state-value and log-policy is popular in the entropy-regularized online RL literature (Nachum et al., 2017; Peters et al., 2010), where the μ is treated as the policy π to be learned. Our own setting is markedly different from these

previous approaches in that μ is the behavior policy and is fixed. Nevertheless, what would happen if we were to parameterize Q-values in this way, i.e, as

$$Q_\theta(s, a) := V_\theta(s) + \log \mu(a|s), \tag{6.4}$$

and then learn via standard TD error minimization equation 6.2? Well, there is an advantage, but also a disadvantage.

First, the main advantage of this formulation is that, in contrast to a Q-function parameterized by its own neural network function approximator, the density $\mu(a|s)$ is well-defined for all actions, and thus Q is more likely to have a well-behaved landscape even though its training may only cover a small subset of the entire action space. Accordingly, the learned policy π, when trained in the standard way to choose actions that maximize the Q-values, is thus encouraged to stay close to μ, without the need for an explicit divergence penalty as in equation 6.1. In practice, knowledge of μ is not explicitly provided, and one usually resorts to behavioral cloning on the offline dataset to approximate μ (Wu et al., 2019). Nevertheless, the advantage of the formulation in equation 6.4 still holds, since even if we train μ on a sparse subset of all possible actions, the fact that it is a normalized probability distribution means that $\mu(a|s)$ assigns low probabilities to actions outside of $\mathcal{D}$. Still, in practice, some density models might fail to generalize to out-of-distribution data Kirichenko et al. (2020). For this reason, we parameterize the approximate density μ as a mixture density model Bishop (1994).

As for the disadvantage, it is clear that the formulation of Q in equation 6.4 is too restrictive. If we use this representation for training a new policy π, the new policy will be limited to copying the behavior policy μ, regardless of V_θ, which will lead to suboptimal performance. In order to address this issue and enable the learned π to generalize beyond

just mimicking μ, we propose replacing the value function $V_\theta(s)$ with a state-action value *offset* function $O_\theta(s, a)$:

$$Q_\theta(s, a) := O_\theta(s, a) + \log \mu(a|s). \tag{6.5}$$

With this representation, one can learn a richer representation of Q-values. However, this parameterization can potentially put us back in the fully-parameterized Q_θ regime of vanilla actor-critic. It is clear that the offset term must be suitably constrained or regularized to find an appropriate middle-ground between the overly-restrictive $V_\theta(s)$ and the fully-parameterized alternative that is liable to extrapolation errors and policy divergence.

To motivate what an appropriate regularization on O_θ should be, we begin by dissecting exactly how the offset term impacts the learned policy. Let's take a closer look at the policy updates in standard continuous-control actor critic (Haarnoja et al., 2018b; Lillicrap et al., 2015). These updates are based on the chain-rule gradient computation below:

$$\nabla_\phi Q_\theta(s, \pi_\phi(s)) = \left[\nabla_a O_\theta(s, a) + \nabla_a \log \mu(a|s)\right]_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s). \tag{6.6}$$

From this expression, it is clear that potential extrapolation issues with the actor arise via the gradient $\nabla_a O_\theta(s, a)$. That is, without appropriate constraints or regularizers on $O_\theta(s, a)$, its gradients might dominate over the gradients of the behavior policy $\nabla_a \log \mu(a|s)$. Therefore, as a way to control the trade-off between over-constraining π to be close to the behavior policy and learning more rich representations of Q-values, we propose using a gradient penalty regularizer of the form $\|\nabla_a O_\theta(s, a)\|^2$. Accordingly,

the full critic optimization objective is as follows:

$$\min_{\theta} J(O_\theta + \log \mu) + \lambda \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\phi(\cdot|s)}} [\|\nabla_a O_\theta(s,a)\|^2], \tag{6.7}$$

where $J(\cdot)$ is defined as in eq. (6.2) and μ is not updated during critic learning. In this objective, λ is a hyperparameter that controls the contribution of the gradient penalty term. Unless otherwise noted, we set $\lambda = 0.1$ as the regularization coefficient.

A keen reader may note that the gradients in eq. (6.6) look similar to gradients of eq. (6.1), with the difference that we take gradients of the offset function instead of the critic function. However, in our setting, the critic loss is substantially different since the offset term plus the logarithm of behavior density learns to predict the *unmodified* Q-values of the training policy instead of Q-values augmented with a KL-divergence term. For example, if the MDP rewards already naturally compel the learned policy to match the behavior policy, the offset term in Fisher-BRC can vanish, whereas the explicit divergence penalty in BRAC will bias the learned Q-values unnecessarily.

6.4.2 Fisher Divergence Derivation

We now show how the same objective in equation 6.7 may be derived from the perspective of Fisher divergence regularization. We begin by introducing the Boltzmann policies – essentially, policies expressed as a Boltzmann distribution with energy function given by a set of Q-values. We then present the Fisher divergence and show how a Fisher divergence regularizer between a Boltzmann policy and the behavior policy reduces to the gradient penalty proposed in equation 6.7. Finally, we elaborate on connections to CQL, which we show can be interpreted as a KL divergence regularization between the Boltzmann policy and the behavior policy. This insight may be of independent interest

since the original derivation of CQL is via a very different route.

Boltzmann Policies For a given Q-value function, the associated Boltzmann policy is given by the following expression:

$$\pi_{ebm}(a|s) := \frac{\exp(Q(s,a))}{\sum \exp(Q(s,a))}. \tag{6.8}$$

In an actor-critic setting, the actor will recover this same policy if an entropy regularizer is added to the actor loss, as is commonly done (Haarnoja et al., 2018b). While some works use this representation of a policy more explicitly (Fox et al., 2015; Haarnoja et al., 2017; Nachum et al., 2017), the main disadvantage is that the normalization term may not be tractable for continuous actions, and so computing the policy distribution π_{ebm} explicitly requires performing computationally expensive numerical integration. Thus, it is more common to only recover the policy via entropy regularization on the actor loss.

Fisher Divergence In order to avoid issues with computing the normalization term, we consider the Fisher divergence, or Fisher information distance Johnson (2004):

$$F(p(\cdot), q(\cdot)) = \mathop{\mathbb{E}}_{x \sim p(\cdot)} \left[\|\nabla_x \log p(x) - \nabla_x \log q(x)\|^2 \right]. \tag{6.9}$$

As one can see from the formulation, in order to compute the Fisher divergence between two distributions, we need only have sampling access to $p(x)$ and the ability to compute $\nabla_x \log p(x), \nabla_x \log q(x)$. Crucially, the computation of either $\nabla_x \log p(x)$ or $\nabla_x \log q(x)$ does not require normalized distributions, since the normalization term (which is a constant with respect to x) disappears from $\log p(x), \log q(x)$ due to the differentiation. Thus, we can avoid computing the normalization term in equation 6.8.

Since the Fisher divergence is amenable to Boltzmann representations of policies, let us consider an optimization objective that consists of a squared TD-loss and a Fisher divergence term between the Boltzmann distribution and the behavior policy μ:

$$\begin{aligned} &J(Q_\theta) + \lambda \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[F\left(\frac{\exp(Q(s,\cdot))}{\sum_a \exp(Q(s,a))}, \mu(\cdot|s) \right) \right] = \\ &J(Q_\theta) + \lambda \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_{emb}(\cdot|s)}} \left[\|\nabla_a \log \mu(a|s) - \nabla_a Q(s,a)\|^2 \right]. \end{aligned} \tag{6.10}$$

The coefficient λ controls the strength of the Fisher divergence term. We can further simplify this objective by using the representation of Q proposed in equation 6.5:

$$J(O_\theta + \log \mu) + \lambda \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_{ebm}(\cdot|s)}} \left[\|\nabla_a O_\theta(s,a)\|^2 \right]. \tag{6.11}$$

The only difference with eq. (6.7) is that actions are sampled from the training policy π_ϕ, while in this case, the actions are sampled from the Bolzman policy π_{ebm}. In practice, sampling from π_{ebm} can be just as computationally expensive as computing the normalization term in eq. (6.3). However, as mentioned earlier, the Boltzmann policy π_{ebm} may be recovered by the actor via incorporating an entropy regularizer in the actor loss. Thus, we propose to train our actor π_ϕ precisely in this manner (which is already popular in model-free actor-critic algorithms (Haarnoja et al., 2018b)), and then use it in eq. (6.11) as a plug-in approximation of π_{ebm}. In this way, we have arrived again at the same objective first defined in section 6.4.1.

The connection of our proposed objective to the Fisher divergence recalls similar quantities in the score matching and energy-based generative model literature. The Fisher divergence is a popular metric in this literature because it avoids an expensive

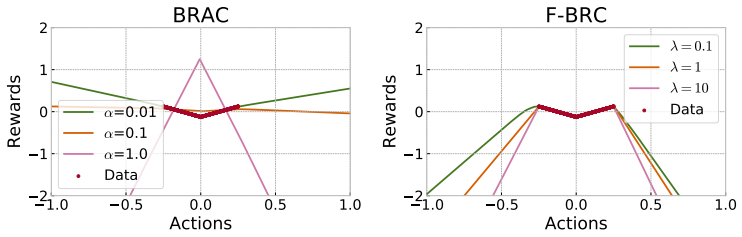


Figure 6.1: The objective landscapes (the regularized critic values) for the policy-induced by the learned critic in BRAC or the parameterized offset critic in Fisher-BRC. The observed actions in the offline data are all within $[-0.25, 0.25]$ and suggest the optimal reward-maximizing actions as $\{-0.25, 0.25\}$. In BRAC (left), we see the landscape is heavily dependent on the choice of KL-divergence coefficient α, and it is easy to either over-regularize (with an optimum around 0.0) or over-extrapolate (with optima far from the observed actions in $[-0.25, 0.25]$). On the other hand, due to the unique parameterization used in Fisher-BRC critic, its corresponding objective landscape correctly predicts the pessimistic reward values and peaks at the modes of the true reward function (right). We also see that Fisher-BRC is more robust to the choice of regularizer coefficient λ.

computation of a log-normalizer, which is necessary for other common divergences (Bao et al., 2020; Lyu, 2012). Moreover, many works in the generative model literature employ gradient penalties, even if they do not explicitly connect to the Fisher divergence. This provides a further empirical advantage to our method, as these gradient penalties – due to their popularity – may be efficiently implemented using many modern machine learning libraries. We note that although we use a soft penalty, one can enforce hard constraints as in Spectral Normalized GANs Miyato et al. (2018), but we leave this for future work.

Due to our method's connection to Fisher Divergence, we dub our method Fisher-BRC (Fisher Behavior Regularized Critic).

Connections to CQL The CQL objective, although originally derived in Kumar et al. (2020) from a very different perspective, can also be motivated as a regularizer on a

	BC	BRAC-v	MBOP	CQL (GitHub)	CQL (Ours)	F-BRC (Ours)
hc-r	30.5	28.1	6.3 ± 4.0	27.1 ± 1.3	20.7 ± 0.6	33.3 ± 1.3
h-r	11.3	12.0	10.8 ± 0.3	10.6 ± 0.1	10.4 ± 0.1	11.3 ± 0.2
w-r	4.1	0.5	8.1 ± 5.5	1.1 ± 2.2	10.0 ± 4.6	1.5 ± 0.7
hc-m	36.1	45.5	44.6 ± 0.8	40.3 ± 0.3	38.9 ± 0.3	41.3 ± 0.3
w-m	6.6	81.3	41.0 ± 29.4	77.3 ± 3.8	69.2 ± 8.3	78.8 ± 1.0
h-m	29.0	32.3	48.8 ± 26.8	42.2 ± 15.5	30.5 ± 0.7	99.4 ± 0.3
hc-e	107.0	-1.1	-	54.4 ± 45.8	103.5 ± 1.3	108.4 ± 0.5
h-e	109.0	3.7	-	67.7 ± 54.7	112.2 ± 0.2	112.3 ± 0.1
w-e	125.7	-0.0	-	84.7 ± 42.7	107.2 ± 3.8	103.0 ± 5.0
hc-m-e	35.8	43.8	105.9 ± 17.8	21.7 ± 6.8	58.6 ± 8.7	93.3 ± 10.2
w-m-e	11.3	-0.3	70.2 ± 36.2	104.0 ± 10.1	104.6 ± 10.4	105.2 ± 3.9
h-m-e	111.9	1.1	55.1 ± 44.3	111.3 ± 2.1	112.4 ± 0.2	112.4 ± 0.3
hc-mix	38.4	45.6	42.3 ± 0.9	44.9 ± 1.1	42.0 ± 1.1	43.2 ± 1.5
h-mix	11.8	0.7	12.4 ± 5.8	31.6 ± 3.6	29.0 ± 0.5	35.6 ± 1.0
w-mix	11.3	-0.3	9.7 ± 5.3	16.8 ± 3.1	16.5 ± 4.9	41.8 ± 7.9

Table 6.1: Comparison of our method (F-BRC) to prior work. The results for BC and BRAC are taken from Fu et al. (2020); the results for MBOP are taken from Argenson and Dulac-Arnold (2020); the results for CQL (GitHub) are taken from the author-provided open-source implementation of (Kumar et al., 2020); and the results for CQL (Ours) are from our own re-implementation of CQL. For all methods we run ourselves, we plot the normalized returns at the end of training (without early stopping) computed over 5 seeds. For every seed we run evaluation for 10 episodes.

Boltzmann policy. In the case of CQL, the regularizer is the common KL-divergence:

$$J(Q_\theta) + \lambda \mathop{\mathbb{E}}_{s \sim \mathcal{D}} \left[D_{KL} \left(\mu(\cdot|s) \middle| \frac{\exp(Q(s, \cdot))}{\sum_a \exp(Q(s, a))} \right) \right]. \tag{6.12}$$

Expanding the KL-Divergence term yields,

$$D_{KL}(\mu(\cdot|s)|\pi_{emb}(\cdot|s)) = \mathop{\mathbb{E}}_{a\sim\mu(\cdot|s)}\left[\log\frac{\mu(a|s)}{\pi_{emb}(a|s)}\right] =$$

$$\mathop{\mathbb{E}}_{a\sim\mu(\cdot|s)}\left[\log\sum_{a}\exp(Q(s,a)) - Q(s,a) + \log\mu(a|s)\right],$$

and this is equivalent to the familiar form of CQL from equation 6.3, since $\log\mu(a|s)$ is a constant with respect to Q.

As mentioned earlier, for continuous distributions, the normalization term in CQL is not tractable and necessitates expensive numerical integration. In CQL, this integration is calculated by sampling from the training policy and importance weighting. Thus, our method enjoys a significant computational advantage. Our use of a novel critic representation and the Fisher divergence allows us to circumvent this practical issue.

We note that in this derivation of CQL, the direction of divergence in eq. (6.12) is switched compared to eq. (6.10) in Fisher-BRC. One may derive a variant of Fisher-BRC using this same direction of the divergence, and this will result in the same expression in eq. (6.7), only that the expectation of the gradient penalty is switched to be over $(s,a) \sim \mathcal{D}$. Anecdotally, we did not observe large empirical differences when training with this alternative objective in initial experiments. However, due to the closer connection to the actor loss when using $a \sim \pi_\phi(\cdot|s)$ (see section 6.4.1), we stick to the formulation originally presented in eq. (6.7).

6.5 Experiments

We present empirical demonstrations of Fisher-BRC in a variety of settings. We start with a simple continuous bandit experiment that illustrates the difference between our method and more common behavior regularization techniques based on explicit divergence penalties. Then we evaluate our method against state-of-the-art offline RL model-based and model-free algorithms on the D4RL benchmark datasets. Finally, we analyze the effect of gradient penalty regularization and provide statistics on the computational advantage of Fisher-BRC compared to CQL.

6.5.1 Implementation Details

Behavior policy We fit the behavior model using a conditional Mixture of Gaussians Bishop (1994) with tanh squashing Haarnoja et al. (2017). We use five mixture components. We train the density model with Adam optimizer Kingma and Ba (2014) for 10^6 steps and starting from learning rate 10^{-3} and decreasing it by ten at $8 \cdot 10^5$ and $9 \cdot 10^5$ gradient update steps. Similar to BRAC, we train the behavior actor with SAC-style entropy regularization with the same target entropy. We parameterize the model as a three-layer MLP with relu activations and 256 hidden units.

actor and critic learning We base our implementation on Soft Actor-Critic Haarnoja et al. (2018b). As in CQL, we do not add entropy to the rewards, and we modify the critic loss to accommodate the additional regularization term. We use default SAC hyperparameters without additional tuning, in contrast to CQL and BRAC, which tune the policy learning rate. Following CQL, we increased the actor's network size and the critic to 3 layer MLP with 256 hidden units.

Survival bonus The linear term used in CQL can be seen as adding a survival bonus for the environments with early termination. We include the derivation in the appendix. Adding a positive constant to the rewards does not affect the optimal policy in infinite horizon MDPs, but in practice, we replace Q-targets for terminal states with 0, which leads to having either a survival bonus or step penalty. For this reason, we add a reward bonus to our implementation as well for a fair comparison. We choose the same value $\lambda_{cql} = 5$ as in CQL.

In particular, one can verify that

$$\begin{aligned}
&\nabla_\theta[-\lambda_{cql} Q_\theta(s,a) + (\gamma Q_{\hat{\theta}}(s',a') + r(s,a) - Q_\theta(s,a))^2] = \\
&-\lambda_{cql} \nabla_\theta Q_\theta(s,a) - (\gamma Q_{\hat{\theta}}(s',a') + r(s,a) - Q_\theta(s,a)) \nabla_\theta Q_\theta(s,a) = \\
&-(\gamma Q_{\hat{\theta}}(s',a') + [r(s,a) + \lambda_{cql}] - Q_\theta(s,a)) \nabla_\theta Q_\theta(s,a) = \\
&\nabla_\theta (\gamma Q_{\hat{\theta}}(s',a') + [r(s,a) + \lambda_{cql}] - Q_\theta(s,a))^2.
\end{aligned}$$

6.5.2 Toy Continuous Bandit Problem

We begin with a simple conceptual demonstration comparing Fisher-BRC to the similar and common alternative of explicit divergence penalties applied to the learned policy. Specifically, we consider the loss in eq. (6.1), which corresponds to the policy update used in BRAC (Wu et al., 2019).

We consider a continuous bandit with one-dimensional action space given by $[-1, 1]$ for this experiment. The rewards are given by

$$r(a) = \begin{cases} |a| - 0.125, & \text{if } a \in [-0.25, 0.25] \\ -\infty, & \text{otherwise.} \end{cases}$$

The offline training dataset is collected by sampling 1000 actions from a uniform distribution, $a \sim \mathcal{U}(-0.25, 0.25)$, and recording the corresponding rewards $r(a)$. Thus, the distribution of actions exhibits inadequate coverage of the entire action space, and the rewards for $a \in [-1, -0.25] \cup [0.25, 1]$ are not observed in the dataset.

Both BRAC and Fisher-BRC require a fitted behavior policy. To do so, we fit a behavior policy $\mu(a)$ parameterized as a Laplace distribution.[1] Subsequently, we fit the critic for BRAC and F-BRC. In BRAC, we fit a critic using mean squared error to match the rewards: $\mathbb{E}_{(a,r)\sim\mathcal{D}}[(R_\theta(a) - r)^2]$. For Fisher-BRC, we use the representation and regularization from eq. (6.7), assuming an initialization of π to $\mathcal{U}$:

$$\mathop{\mathbb{E}}_{(a,r)\sim\mathcal{D}}[(O_\theta(a) + \log \mu(a) - r)^2] + \lambda \mathop{\mathbb{E}}_{a_{reg}\sim\mathcal{U}(-1,1)} \|\nabla_a O_\theta(a_{reg})\|^2.$$

These critics then determine the objective landscape for the learned policy. We plot these landscapes in fig. 6.1. Specifically, we plot $R_\theta(a) + \alpha \log \mu(a)$ for BRAC and $O_\theta(a) + \log \mu(a)$ for Fisher-BRC for a variety of choices of α and λ. Recall that we train the policy will to choose actions that maximize these values. Thus, ideally, these objective landscapes should possess optima around the globally optimal actions $\{-0.25, 0.25\}$.

We observe that it is hard to pick the KL-coefficient α for the policy loss landscape induced by BRAC to avoid either over-generalizing (with optima outside of $[-0.25, 0.25]$) or over regularizing (with optima far from the optimal actions $\{-0.25, 0.25\}$); see fig. 6.1, left.

On the other hand, Fisher-BRC correctly constrains the learned value function – in

[1] Our choice of a Laplace parameterization is to match the absolute value appearing in the definition of rewards $r(a)$. If a Gaussian parameterization is used, then the rewards may be modified to a quadratic function to achieve the same result.

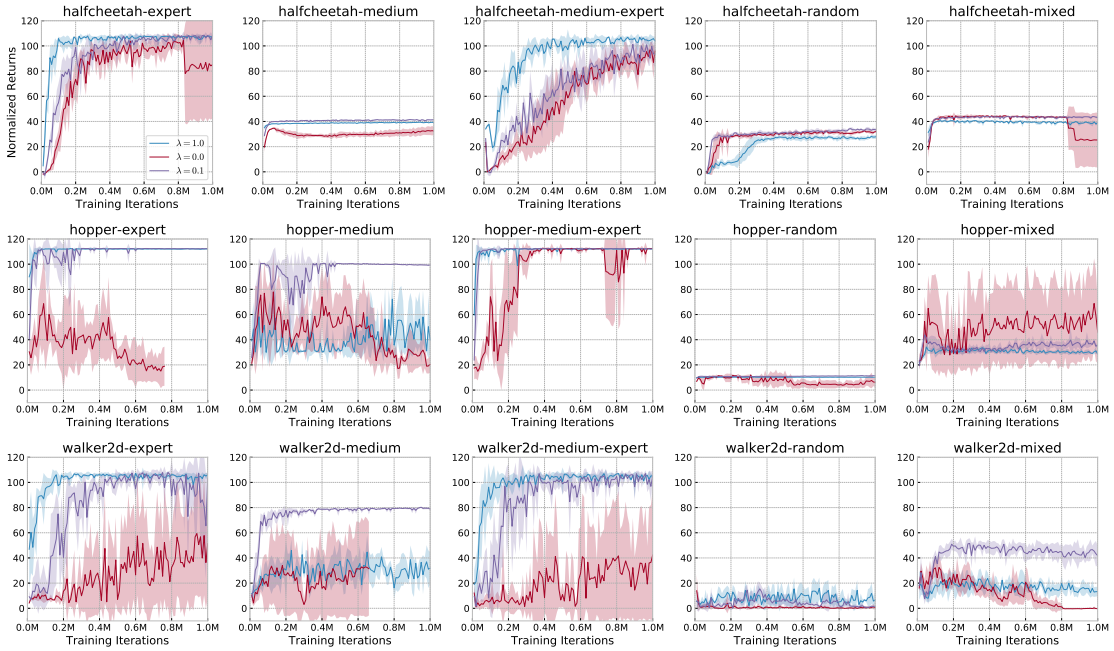


Figure 6.2: Performance of F-BRC for different values of the gradient penalty coefficient. A larger value, $\lambda = 1$, over-constraints the learned policy to stay close to the behavior policy. This leads to more stable performance on expert datasets, where the behavior policy is near-optimal, but worse performance on medium datasets. Without the regularization ($\lambda = 0.0$) Fisher-BRC collapses on most of these tasks; when the plot is cutoff, it means at least one of the seeds produced NaN values in training.

fact, the value function is unmodified for in-distribution samples – and is robust to the choice of the regularization hyperparameter (fig. 6.1, right).

6.5.3 Deep Offline RL Benchmarks

We continue to present Fisher-BRC in more complex environments. We compare our method to prior work on the OpenAI Gym MuJoCo tasks using D4RL datasets (Fu et al., 2020). We consider the following baselines: BRAC-vp and BRAC-pr (Wu et al., 2019), due to a similar policy learning objective, MBOP Argenson and Dulac-Arnold (2020), the state-of-state in offline model-based reinforcement learning, and CQL (Kumar et al., 2020), due to a similar connection with energy-based learning. Our implementation

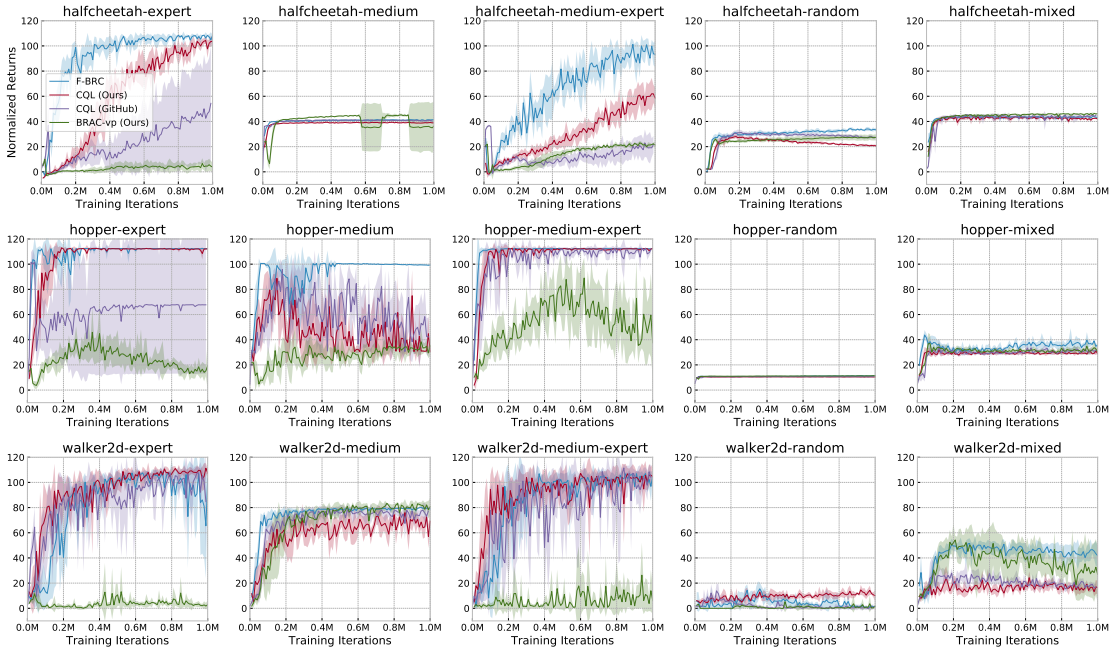


Figure 6.3: We compare F-BRC against prior methods in terms of convergence speed with respect to gradient updates steps. We see that Fisher-BRC enjoys better final performance and faster convergence in most tasks compared to BRAC and CQL.

for Fisher-BRC follows the standard SAC implementation, only that we use a 3-layer network as in CQL. Additional implementation details are in the appendix.

Overall performance We present our method's results and all considered baselines in table 6.1. Our method performs comparably or surpasses prior work on most of the tasks. Notably, many of the baseline algorithms exhibit inconsistent performance across the tasks, achieving good performance on some tasks while poor performance on the others. In contrast, our Fisher-BRC exhibits consistent and good performance across almost all tasks.

Effect of gradient penalty As a way of investigating the effect of gradient penalty vs. the offset parameterization in Fisher-BRC, we evaluate gradient penalty values.

We present results of $\lambda \in \{0.0, 0.1, 1.0\}$ in fig. 6.2. We note that the gradient penalty is an essential component of Fisher-BRC since when $\lambda = 0.0$ performance degrades dramatically. On the other hand, when λ is set too high ($\lambda = 1.0$), we see that the learned policy is over-constrained; i.e., we see that performance on the expert datasets is improved while performance is limited on the medium datasets since the behavior policy in these datasets is highly sub-optimal. This is expected since a high λ corresponds to a significant gradient penalty on the regularization on the offset, compelling the offset to be near-constant with respect to actions.

Critic parametrization We also evaluate the effect of gradient penalty on standard Soft Actor-Critic without the critic representation introduced in this paper.

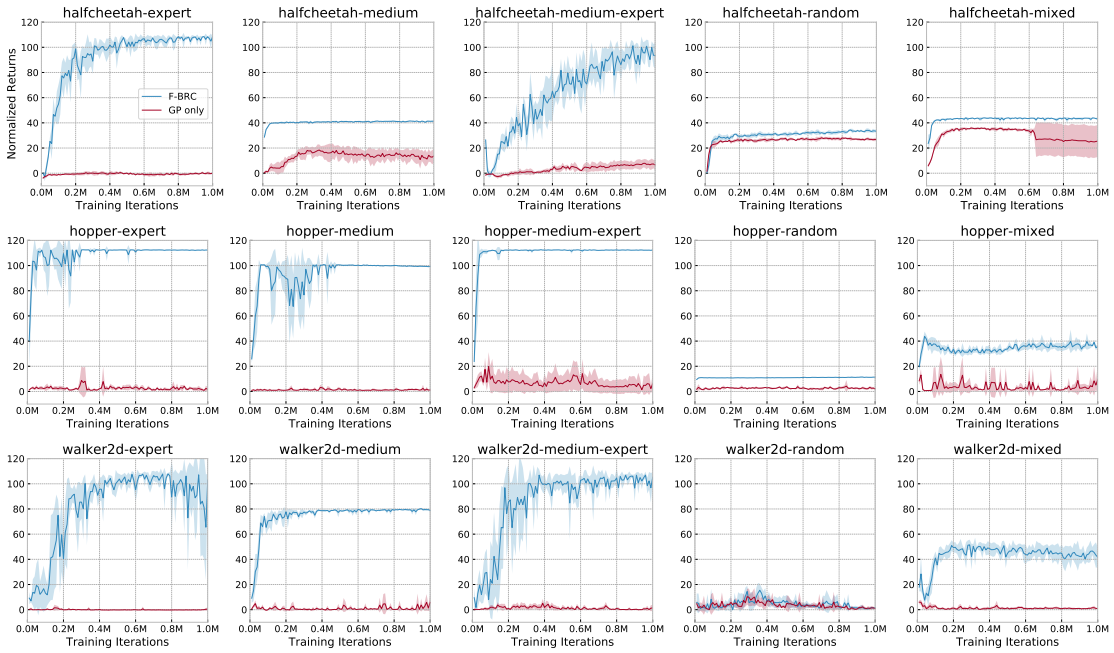


Figure 6.4: Performance of F-BRC without our critic representation. Without the critic representation, the gradient penalty term alone fails to improve performance of the underlying reinforcement learning algorithm on the offline datasets.

Convergence speed We also evaluate our method's wall-clock training time compared with CQL, which demonstrates comparable policy performance on the D4RL tasks but significantly different computational efficiency. The total training time for 1 million steps for Fisher-BRC is 1.4 hours of behavioral cloning pre-training followed by 6.2 hours of policy training. CQL's total training time is 16.3 hours (which does not require pre-training of a behavior density policy). Our method converges faster not only in terms of gradient updates (see fig. 6.3), but it is also computationally faster due to omitting the expensive numerical integration to compute the log-sum-exp term of CQL. These experiments were carried out on a Google cloud instance containing an AMD EPYC 7B12 CPU at 2.25GHz (using 8 of 64 available cores) and 32GB of RAM.

6.6 Conclusions

We have introduced Fisher-BRC, a simple critic representation and regularization technique for offline reinforcement learning. Our derivations highlight connections between our training objective and Fisher divergence regularization from score matching and energy-based model literature. Our method is easy to implement and highly performant. Compared to existing offline RL algorithms, Fisher-BRC exhibits better and more consistent performance across various domains.

Chapter 7

Conclusion

This thesis aims to explore different ways to improve the sample efficiency of imitation and reinforcement learning methods. In particular, we focused on several topics: 1) off-policy and offline imitation learning; 2) off-policy learning from pixels; and 3) offline reinforcement learning. We introduced a novel algorithm or technique for each topic that improves sample efficiency of convergence speed compared to the previous state-of-the-art.

In chapter 3, we investigate the problem of sample inefficiency of the family of imitation learning methods (Ho and Ermon, 2016) based on adversarial learning (Goodfellow et al., 2014). We introduce Discriminator-Actor-Critic, an off-policy approach for imitation learning (Kostrikov et al., 2019), we use off-policy RL for policy learning, and we introduce a simple, practical approach for off-policy discriminator training. Instead of training the discriminator with on-policy samples, we sample from the replay buffer collected during training. We also identify a problem with reward bias associated with these methods and describe how to modify the MDP to handle learned rewards properly. The off-policy training component of our approach improves the sample efficiency of GAIL

by order of magnitude. At the same time, correct handling of absorbing states eliminates the need for incorporating the task-specific information into the reward function.

In chapter 4, we focus on a completely offline approach to imitation learning. First, inspired by DualDICE Nachum et al. (2019b) we derive a principled offline formulation of the KL-divergence between occupancy measures. In contrast to Discriminator-Actor-Critic (DAC) described in chapter 3, ValueDICE directly minimizes the objective of interest instead of minimizing a surrogate objective as in DAC. Then, we adapt this formulation for practical applications in imitation learning. In particular, we provide another formulation of the KL-divergence matching objective that can incorporate additional samples from an arbitrary policy. We demonstrate our approach's performance on a synthetic task and a standard set of continuous control tasks. Our approach outperforms the prior state-of-the-art, including behavioral cloning in the completely offline regime. We dub our method as ValueDICE. ValueDICE is the first approach that omits learning rewards and learns a value function that corresponds to these rewards directly to the best of our knowledge. The implicit divergence term can also be used to augment the critic in offline reinforcement learning, which we leave for future work.

In chapter 5, we introduce DrQ, an image augmentation technique that can be plugged into any off-policy RL algorithm. Model-free off-policy RL methods such as Soft-Actor-Critic and Deep Deterministic Policy Gradients achieve state-of-the-art performance on tasks where inputs represent low dimensional vectors. At the same time, these algorithms fail to outperform model-based and auxiliary loss methods when they are trained from pixel inputs. We identify that due to off-policy training from replay buffers, model-free methods suffer from overfitting. By leveraging image perturbations commonly used in Computer Vision, DrQ overcomes the issue of overfitting and achieves the state-of-the-art performance of several benchmarks for continuous and discrete control in terms of

sample efficiency. We further improve the method by averaging several perturbations for computing the targets and reusing these targets between different augmentations of inputs. The generality and simplicity of the method can enable a wide variety of practical applications of reinforcement learning. Moreover, the same image augmentation techniques can be applied to Discriminator-Actor-Critic (chapter 3) and ValueDICE (chapter 4). Both of these directions are exciting avenues for future work.

In chapter 6, we focus on Offline Reinforcement Learning and introduce Fisher-BRC, a method for critic regularization in model-free reinforcement learning. We argue that end-to-end training of a critic that incorporates behavior constraints improves performance and stability of training compared to methods that constrain the training policy only. This work describes two contributions: first, we propose a simple reparametrization of critic that allows us to rely on a pre-trained density model simplifying subsequent training. Second, we demonstrate that using this reparametrization allows us to express Fisher divergence between two distributions as a simple gradient penalty term. Computing gradient penalty is computationally less expensive than explicitly computing the normalizer used in other methods based on Energy-Based learning. Note that in contrast to ValueDICE (algorithm 4.1), Fisher-BRC enforced a KL-constraint only on policies instead of occupancy measures. Thus, Fisher-BRC can be further extended by incorporating a state-dependent term similarly to ValueDICE (algorithm 4.1). We can also explore applying Fisher-BRC to pixel inputs using the techniques from DrQ (chapter 5) since both are based on the same underlying reinforcement learning algorithm, Soft-Actor-Critic. We recommend these directions for future work since both can further expand the variety of reinforcement learning applications.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

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

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018). Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*.

Ahmed, Z., Roux, N. L., Norouzi, M., and Schuurmans, D. (2018). Understanding the impact of entropy on policy optimization. *arXiv preprint arXiv:1811.11214*.

Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J.,

Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2018). Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Argenson, A. and Dulac-Arnold, G. (2020). Model-based offline planning. *arXiv preprint arXiv:2008.05556*.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv e-prints*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bain, M. and Sammut, C. (1995). A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2017). Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.

Bao, F., Xu, K., Li, C., Hong, L., Zhu, J., and Zhang, B. (2020). Variational (gradient) estimate of the score function in energy-based latent variable models. *arXiv preprint arXiv:2010.08258*.

Baram, N., Anschel, O., Caspi, I., and Mannor, S. (2017). End-to-end differentiable adversarial imitation learning. In *International Conference on Machine Learning*, pages 390–399.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributional policy gradients. In *International Conference on Learning Representations*.

Becker, S. and Hinton, G. E. (1992). Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*.

Belghazi, M. I., Baratin, A., Rajeswar, S., Ozair, S., Bengio, Y., Courville, A., and Hjelm, R. D. (2018). Mine: mutual information neural estimation. *arXiv preprint arXiv:1801.04062*.

Bellman, R. (1957). A markovian decision process. *Indiana Univ. Math. J.*

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition.

Bishop, C. M. (1994). Mixture density networks. Technical Report.

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*.

Brock, A., Donahue, J., and Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for

contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Ciregan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649.

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2019). Leveraging procedural generation to benchmark reinforcement learning. *arXiv:1912.01588*.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2018). Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*.

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*.

Donsker, M. D. and Varadhan, S. S. (1983). Asymptotic evaluation of certain markov process expectations for large time. iv. *Communications on Pure and Applied Mathematics*, 36(2):183–212.

Dosovitskiy, A., Fischer, P., Springenberg, J. T., Riedmiller, M., and Brox, T. (2016).

Discriminative unsupervised feature learning with exemplar convolutional neural networks. *TPAMI*.

Dwibedi, D., Tompson, J., Lynch, C., and Sermanet, P. (2018). Learning actionable representations from visual observations. *CoRR*.

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*.

Farahmand, A., Ghavamzadeh, M., Szepesvari, C., and Manor, S. (2008). Regularized policy iteration. In *NIPS*.

Farebrother, J., Machado, M. C., and Bowling, M. (2018). Generalization and regularization in dqn. *arXiv abs/1810.00123*.

Finn, C., Christiano, P., Abbeel, P., and Levine, S. (2016a). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *NIPS Workshop on Adversarial Training*.

Finn, C., Levine, S., and Abbeel, P. (2016b). Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58. PMLR.

Fox, R., Pakman, A., and Tishby, N. (2015). Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*.

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

Fu, J., Luo, K., and Levine, S. (2017). Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062.

Fujimoto, S., van Hoof, H., and Meger, D. (2018a). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018b). Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmassan, Stockholm, Sweden, July 10-15, 2018*.

Ghasemipour, S. K. S., Schuurmans, D., and Gu, S. S. (2021). Emaq: Expected-max q-learning operator for simple yet effective offline and online rl.

Ghasemipour, S. K. S., Zemel, R., and Gu, S. (2020). A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR.

Gonzalez-Garcia, A., van de Weijer, J., and Bengio, Y. (2018). Image-to-image translation for cross-domain disentanglement.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

Gupta, A., Murali, A., Gandhi, D. P., and Pinto, L. (2018). Robot learning in homes: Improving generalization and reducing dataset bias. In *Advances in Neural Information Processing Systems*.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. (2018a). Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 1851–1860. PMLR.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., a nd Pieter Abbeel, A. G., and Levine, S. (2018b). Soft actor-critic algorithms and applications. *CoRR*.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

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*.

Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. J. (2017). Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 1235–1245.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2019). Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*.

Heess, N., Silver, D., and Teh, Y. W. (2013). Actor-critic reinforcement learning with energy-based policies. In Deisenroth, M. P., Szepesvári, C., and Peters, J., editors, *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 45–58, Edinburgh, Scotland. PMLR.

Hénaff, O. J., Srinivas, A., Fauw, J. D., Razavi, A., Doersch, C., Eslami, S. M. A., and van den Oord, A. (2019). Data-efficient image recognition with contrastive predictive coding. *CoRR*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. *Thirty-Second AAAI Conference On Artificial Intelligence (AAAI)*.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., et al. (2017). Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012a). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573.

Jaderberg, M., Mnih, V., Czarnecki, W., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. *International Conference on Learning Representations*.

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*.

Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J. M., Turner, R. E., and Eck, D. (2017). Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. In *International Conference on Machine Learning*, pages 1645–1654. PMLR.

Johnson, O. (2004). *Information theory and the central limit theorem*. World Scientific.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*.

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

Kang, B., Jie, Z., and Feng, J. (2018). Policy optimization with demonstrations. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2469–2478. PMLR.

Kappen, H. J., Gómez, V., and Opper, M. (2012). Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182.

Ke, L., Barnes, M., Sun, W., Lee, G., Choudhury, S., and Srinivasa, S. (2019). Imitation learning as f-divergence minimization. *arXiv preprint arXiv:1905.12888*.

Kielak, K. P. (2020). Do recent advancements in model-based deep reinforcement learning really improve data efficiency? *openreview*.

Kim, K.-E. and Park, H. S. (2018). Imitation learning via kernel mean embedding. *AAAI*, 32(1).

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kirichenko, P., Izmailov, P., and Wilson, A. G. (2020). Why normalizing flows fail to detect out-of-distribution data.

Kostrikov, I., Agrawal, K. K., Dwibedi, D., Levine, S., and Tompson, J. (2019). Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *International Conference on Learning Representations*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.

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*.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*.

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

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. (2020). Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*.

Le, H. M., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and Daumé III, H. (2018). Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*.

Lee, A. X., Nagabandi, A., Abbeel, P., and Levine, S. (2019). Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv e-prints*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Li, Y., Song, J., and Ermon, S. (2017). Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3812–3822.

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. *CoRR*.

Liu, Z., Li, X., Kang, B., and Darrell, T. (2019). Regularization matters in policy optimization. *arXiv abs/1910.09191*.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2017). Are gans created equal? a large-scale study. *arXiv preprint arXiv:1711.10337*.

Lyu, S. (2012). Interpretation and generalization of score matching. *arXiv preprint arXiv:1205.2629*.

Matsushima, T., Furuta, H., Matsuo, Y., Nachum, O., and Gu, S. (2020). Deployment-efficient reinforcement learning via model-based offline optimization. *arXiv preprint arXiv:2006.03647*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

Misra, I. and van der Maaten, L. (2019). Self-supervised learning of pretext-invariant representations. *arXiv:1912.01991*.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*.

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

Muller, U., Ben, J., Cosatto, E., Flepp, B., and Cun, Y. L. (2006). Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746.

Nachum, O., Ahn, M., Ponte, H., Gu, S., and Kumar, V. (2019a). Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*.

Nachum, O., Chow, Y., Dai, B., and Li, L. (2019b). Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *arXiv preprint arXiv:1906.04733*.

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

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2017). Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Peters, J., Mulling, K., and Altun, Y. (2010). Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

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

Puterman, M. L. (2014). *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

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

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

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

Sasaki, F., Yohira, T., and Kawaguchi, A. (2019). Sample efficient imitation learning for continuous control. In *International Conference on Learning Representations*.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv e-prints*.

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. PMLR.

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

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. (2018). Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Slaoui, R. B., Clements, W. R., Foerster, J. N., and Toth, S. (2019). Robust visual domain randomization for reinforcement learning. *arXiv abs/1910.10537*.

Srinivas, A., Laskin, M., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction*. MIT press.

Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039. ACM.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Thomas, P. S. (2015). *Safe reinforcement learning*. PhD thesis, University of Massachusetts Libraries.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE.

Torabi, F., Warnell, G., and Stone, P. (2018). Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*.

van Amersfoort, J., Smith, L., Teh, Y. W., and Gal, Y. (2020). Uncertainty estimation using a single deep deterministic neural network.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *arXiv e-prints*.

van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*.

Vecerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. A. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR, abs/1707.08817*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, Z., Merel, J. S., Reed, S. E., de Freitas, N., Wayne, G., and Heess, N. (2017). Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pages 5320–5329.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Williams, R. J. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*.

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

Yan, X., Choromanski, K., Boots, B., and Sindhwani, V. (2017). Manifold regularization for kernelized lstd. *arXiv abs/1710.05387*.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., and Fergus, R. (2019). Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *arXiv preprint arXiv:1910.10897*.

Zhu, Y., Wang, Z., Merel, J., Rusu, A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., et al. (2018). Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*.