From Adversarial Imitation Learning
to Robust Batch Imitation Learning

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37364689">https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37364689</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
From Adversarial Imitation Learning to Robust Batch Imitation Learning

A THESIS PRESENTED
BY
Yecheng J. Ma
TO
The School of Engineering and Applied Sciences
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Bachelor of Arts (Honors)
in the subject of
Computer Science

Harvard University
Cambridge, Massachusetts
April 2020
From Adversarial Imitation Learning to Robust Batch Imitation Learning

Abstract

Imitation learning (IL) aims to learn a behavior policy through imitating the behavior of an expert. While successfully achieving high performance in various domains, IL lacks an established set of evaluation metrics that makes comparing algorithms and identifying their shortcomings difficult. This thesis proposes a suite of evaluation metrics for imitation learning, and benchmarks Behavior Cloning (BC) and Generative Adversarial Imitation Learning (GAIL), two baseline IL algorithms. Our results challenge the consensus that GAIL is favorable to BC, and argue that any perceived gain is due to a non-standard training methodology employed in prior work. In addition, these evaluations discover a shortcoming in both algorithms that has not been adequately addressed. That is, they are susceptible to expert data that consists of a mixture of optimal and degraded trajectories. Due to the noisy nature of expert data, this significantly hampers the usability of IL in the real-world. Building on recent insights from batch reinforcement learning (BIL) as well as self-supervised reward learning, I propose and study a novel batch imitation learning algorithm, Disagreement-Regularized Batch-Constrained-Q Imitation Learning (DRBIL), which learns without any interaction with the environment and is robust to expert data degradation. These properties ensure that DRBIL can learn a good policy without the agent taking risky actions or overfitting to degraded expert trajectories. I instantiate DRBIL in MuJoCo domains and demonstrate state-of-art IL performance as well as robustness to data degradation. Together, this thesis takes an important step forward in making IL rigorous and suggests a new BIL framework that is widely adaptable and satisfies critical safety desiderata.
## Contents

**Listing of figures**

<table>
<thead>
<tr>
<th>1</th>
<th>INTRODUCTION</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Related Work</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>IMITATION LEARNING PRELIMINARIES</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Reinforcement Learning</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Imitation Learning</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Imitation Learning Applications</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Discussion</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>ON EVALUATING IMITATION LEARNING ALGORITHMS</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Evaluation Metrics in Existing Work</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>A Set of Imitation Learning Evaluation Metrics</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Discussion</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>BENCHMARKING BC &amp; GAIL</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>General Experimental Setups</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Benchmark Results</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Summary of Results</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Discussion</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>FROM BATCH REINFORCEMENT LEARNING TO BATCH IMITATION LEARNING</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Batch Reinforcement Learning Preliminaries</td>
<td>42</td>
</tr>
<tr>
<td>5.2</td>
<td>Batch-Constrained Deep Q-Learning</td>
<td>44</td>
</tr>
<tr>
<td>5.3</td>
<td>Discussion</td>
<td>46</td>
</tr>
</tbody>
</table>

<p>| 6  | DISAGREEMENT REGULARIZED BATCH-CONSTRAINED-Q IMITATION LEARNING | 47 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Self-Supervised Reward Function for Batch Imitation Learning</td>
<td>47</td>
</tr>
<tr>
<td>6.2 The Full Algorithm</td>
<td>50</td>
</tr>
<tr>
<td>6.3 Results</td>
<td>50</td>
</tr>
<tr>
<td>6.4 Reward Agnostic Imitation Learning</td>
<td>58</td>
</tr>
<tr>
<td>7 DISCUSSIONS &amp; FUTURE DIRECTIONS</td>
<td>61</td>
</tr>
<tr>
<td>7.1 Takeaways and Future Work</td>
<td>61</td>
</tr>
<tr>
<td>8 CONCLUSION</td>
<td>64</td>
</tr>
<tr>
<td>A ENVIRONMENT SPECIFICATIONS</td>
<td>65</td>
</tr>
<tr>
<td>A.1 Mujoco</td>
<td>65</td>
</tr>
<tr>
<td>B EXPERT POLICY DETAILS</td>
<td>67</td>
</tr>
<tr>
<td>B.1 Mujoco Environments</td>
<td>67</td>
</tr>
<tr>
<td>B.2 Expert Trajectories</td>
<td>69</td>
</tr>
<tr>
<td>C IMITATION LEARNING MODELS IMPLEMENTATION DETAILS</td>
<td>71</td>
</tr>
<tr>
<td>C.1 Hardware Specification</td>
<td>71</td>
</tr>
<tr>
<td>C.2 Behavior Cloning (Mujoco)</td>
<td>71</td>
</tr>
<tr>
<td>C.3 GAIL</td>
<td>71</td>
</tr>
<tr>
<td>C.4 DRBCQ</td>
<td>73</td>
</tr>
<tr>
<td>D ADDITIONAL RESULTS</td>
<td>74</td>
</tr>
<tr>
<td>D.1 Hopper</td>
<td>75</td>
</tr>
<tr>
<td>D.2 Humanoid</td>
<td>76</td>
</tr>
<tr>
<td>E MISSING BACKGROUND</td>
<td>77</td>
</tr>
<tr>
<td>E.1 Proximal Policy Optimization (PPO)</td>
<td>77</td>
</tr>
<tr>
<td>E.2 Variational Auto-Encoder (VAE)</td>
<td>77</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>79</td>
</tr>
</tbody>
</table>
Listing of figures

2.2.1 GAIL follows a 3-step procedure: 1. The imitator collects trajectories. 2. The discriminator update itself using both expert and imitator trajectories. 3. The discriminator sends signal to the imitator/generator to improve its policy via RL. ....................................................... 12

2.3.1 The first neural network imitation learning based autonomous driving architecture introduced in Pomerleau [1989] ................................................................. 15

2.3.2 OpenAI incorporated imitation learning to develop its Rubik’s cube solving robot arm [Akkaya et al., 2019] ................................................................. 16

3.1.1 Performance of learned policies shown in the original GAIL paper [Ho and Ermon, 2016]. 18

3.1.2 TREE-PASSING: LEFT and RIGHT achieve the same goal but qualitatively differ. ... 20

4.1.1 A Walker2d Agent Walking. ................................................................. 29

4.2.1 BC & GAIL Best Performance on Walker2d-v2 ........................................ 31

4.2.2 BC & GAIL Running Performance on Walker2d-v2 .................................. 33

4.2.3 Comparing individual runs of GAIL models in Walker2d. ......................... 34

4.2.4 Average Stepwise Likelihood over 5 Walker2d Expert Trajectories ............. 36

4.2.5 Histograms of Cumulative Rewards and Durations of Expert and Imitator Trajectories in Walker2d-v2. ................................................................. 37

4.2.6 Walker2d Noisy Actions (NA) Evaluations ............................................. 38

6.1.1 Poorer trajectories incur higher Uncertainty Cost. ..................................... 49

6.3.1 DRBIL vs. BCQ Mujoco Running Performances. ..................................... 52

6.3.2 BC & DRBIL Walker2d Running Performances ..................................... 54

6.3.3 Histograms of Cumulative Rewards and Durations of Expert and Imitator Trajectories in Walker2d-v2. ................................................................. 55

6.3.4 Walker2d Noisy Actions (NA) Evaluations ............................................. 56

6.4.1 RBIL (“BCQ - random”) (green) achieves competitive performance on the Mujoco environments. ................................................................. 60

6
A.1.1 Illustration of Hopper, Walker, and HalfCheetah tasks. Figure courtesy of [Duan et al., 2016] ............................... 66

B.1.1 PPO expert training curves in Mujoco environments ........................................ 69
B.2.1 Mujoco Expert Trajectory Performance Distribution ........................................ 70

D.1.1 BC & DRBCQ Hopper Best Performances ....................................................... 75
D.1.2 BC & DRBCQ Hopper Running Performances .................................................. 75
D.1.3 BC & DRBCQ Hopper Noisy Actions ............................................................... 75
D.2.1 BC & DRBCQ Humanoid Best Performances .................................................... 76
D.2.2 BC & DRBCQ Humanoid Running Performances ............................................. 76
D.2.3 BC & DRBCQ Humanoid Noisy Actions, $\epsilon = 0$ ........................................ 76
Acknowledgments

First and foremost, I would like to express my sincerest gratitude to my thesis advisor Professor David C. Parkes for his continuous support and feedback throughout my thesis, and for inspiring me with his kindness, resourcefulness, and scholarship. As I soon embark on my graduate school journey, I am tremendously grateful to have David as a role model whom I will always look up to.

I would like to thank Professor Susan Murphy and Finale Doshi-Velez for kindly offering to be my thesis readers, for writing me letters of recommendation, and for introducing me to the world of machine learning and research through CS181, CS282, STAT 111, and STAT 234.

I would also like to express my gratitude to other Harvard professors and graduate student mentors who have offered me invaluable advices the past four years and whom I am so fortunate to have known, including Joe Blizstein, Natesh Pillai, Elisabeth Spelke, Na Li, William Kirby, Kageyama Sensei, Walter Dempsey, Goran Radanovic, Momchil Tomov, Lucy Lai, and many others.

I am deeply humbled and grateful for my friends at Harvard, who have supported me throughout this journey, and continue to amaze and inspire me, even at this time when our times on campus have been unfortunately cut short. Thanks to Nina Chen for being my rock and always motivating me to be my best, and Christine Jou for spending countless hours working on thesis with me and being the amazing human being she is. Besides, I am fond of the camaraderie of Michelle Aye, Sharleen Loh, Evelyn Mo, Ani Suresh, Michael Zhou, Andrew Rittenhouse, Karen Chen, and many others.

I would also like to thank Suriya Babu, Aditya Babu, Sam Choi, and Rahul Chopra, for always sticking by my side despite physical distances.

Finally, I would like to thank my mother, father, and sister for their unconditional support and love. It is only with their sacrifices that I am able to pursue my American dream.
Acquiring knowledge is a form of imitation.
Jiddu Krishnamurti

1

Introduction

Learning to act intelligently in an uncertain world has always been seen as one of the hallmarks in artificial intelligence (AI) and machine learning (ML) [Minsky, 1961] [LeCun et al., 2015]. The modern approach takes the form of reinforcement learning (RL) [Sutton and Barto, 2018], by which an agent acquires intelligent behavior through “trial and error” - taking actions, observing their rewards, and discriminating the good actions from the bad ones.

With the recent breakthroughs in artificial neural network and compute processing power, which allow wide and deep neural network to be efficiently built and computed, the field of deep learning (DL) has flourished and accomplished many impressive achievements, ranging from image classification to speech recognition [Krizhevsky et al., 2012] [Hinton et al., 2012]. The main feature of deep learning is its ability to automatically construct effective features and approximate non-linear complex statistical patterns [LeCun et al., 2015]. These two characteristics allow DL to scale machine learning methods to high-dimensional and data-intensive problems that were previously infeasible with hand-crafted features.

Deep learning combined with reinforcement learning, or in short deep reinforcement learning (DRL), has also rose to prominence. To date, DRL has achieved impressive human and superhuman levels of mastery in various domains, including Atari games Mnih et al. [2015], control and robot manipulation [Schulman et al., 2015a, 2017], Go and other strategic boardgames [Silver et al., 2016; Schrittwieser et al., 2019], as
well as modern complex, multi-agent strategic video games such as Dota [Berner et al., 2019]. All these results seem to suggest that DRL is ready to transform our modern society by automating manual tasks that can be more efficiently and correctly performed by an AI trained with RL.

Despite the glamorous and dazzling display of recent successes, the reality paints a much less rosy picture for DRL. Its juggernaut ability in solving simulated environments often does not apply to real-world environments. Contrast to simulated domains such as video games in which rewards are easy to define, real-world domains do not assume a well-defined reward function that allows RL to learn a good policy. Consider the task of learning a good driving policy. While a good human driver is able to drive safely on roads, she may not be able to mathematically formulate a reward function that she is implicitly optimizing for by driving safely. Without a good reward function, RL is ill-posed to solve the autonomous driving problem [Codevilla et al., 2019].

In challenging real-world domains such as autonomous driving, imitation learning (IL), by which an agent improves its policy by mimicking an expert, has shown great promise [Ross and Bagnell, 2010]. IL has proven to be effective and is in fact quite natural in domains that do not assume a well-defined reward function. For example, in the driving domain, while defining a reward function is difficult, learning a good policy is achievable by directly imitating actions taken by an expert driver, effectively bypassing the need for RL. Additionally, even if rewards from the environment are available, IL proves to be a useful technique to pre-train a RL model to enable more efficient training Hester et al. [2018].

While there has been significant progress in IL [Ross et al., 2011] [Ho and Ermon, 2016] [Brantley et al., 2020], no consensus on how to evaluate IL algorithms and how and what to report in results has been reached [Osa et al., 2018]. For example, many papers only report the final performance of the algorithm with respect to the number of expert trajectories given [Reddy et al., 2019; Ho and Ermon, 2016], while some papers are more concerned with the efficiency of the algorithm and provide results in terms of the sample complexity of environmental interactions the algorithm requires to obtain a good performance [Brantley et al., 2020; Blondé and Kalousis, 2018]. Finally, in some domains such as Human-AI coordination (which we discuss in detail next chapter), the purpose of imitation learning is to simply mimic the behavior of the given trajectories [Carroll et al., 2019]; yet, no quantitative evaluation of how well the model mimics the expert behavior is proposed or presented.

This lack of common evaluation metrics makes comparing different IL algorithms difficult. Consequently, it is difficult to identify key shortcomings in existing methods and then propose new methods that address these issues. Furthermore, this oversight risks putting IL into the crisis of reproducibility and validity that DRL is current facing. DRL algorithms are known to be notoriously difficult to tune and
exhibit high variance in performance [Hester et al., 2018]. A close examination of state-of-art algorithms even reveals that the superior performance of Proximal Policy Optimization (PPO) [Schulman et al., 2017] compared to Trust Region Policy Optimization (TRPO) [Schulman et al., 2015a] comes entirely from implementation level optimizations as opposed to PPO’s claimed algorithmic improvements [Engstrom et al., 2020] [Ilyas et al., 2020]. In fact, the gradient stabilizing heuristics deployed by PPO are shown to be flawed in theory, and carry no improvement in practice. With the validity of a supposedly state-of-art and widely adopted DRL algorithm being directly challenged, it is natural to wonder to what extent is the recent advances in DRL in general are robust and trustworthy? Likewise, the same question can be proposed for IL, whose recent advances build primarily on top of advancements in DRL [Ho and Ermon, 2016] [Reddy et al., 2019] [Brantley et al., 2020]. Given the accelerated pace at which the field of IL is producing new algorithms and results, it has become imperative to validate existing IL results by proposing rigorous evaluation metrics and putting current algorithms under the test.

This thesis aims to provide a study in benchmarking IL algorithms by proposing evaluation metrics that evaluate IL algorithms along three important and practical desiderata: performance, imitation, and robustness. In particular, I evaluate two baseline IL algorithms: Behavior Cloning (BC) [Pomerleau, 1989], and Generative Adversarial Imitation Learning (GAIL) [Ho and Ermon, 2016]. BC casts IL as a supervised-learning problem and is a batch method, meaning that it learns a policy off-line without any interaction with the environment. On the other hand, GAIL trains a policy adversarially by interacting with the environment and discriminating expert trajectories against trajectories from its own interactions. In Chapter 2, we review these algorithms in much detail.

The evaluations of BC and GAIL yield new insights about them and reveal a common failure mode that is currently understudied. First, the common view [Ho and Ermon, 2016] [Sasaki et al., 2018] that GAIL is superior than BC is not supported by the evaluations; GAIL’s sample inefficiency coupled with its poor imitation quality makes it not suitable to be used in some practical situations. In contrast, BC trains significantly quicker and obtains excellent performance if the optimality of expert trajectories is assumed. This assumption naturally leads to our second finding that both BC and GAIL are vulnerable to unreliable expert trajectories - unreliable in the sense that when the expert dataset is a mixture of expert trajectories and degraded expert trajectories, both algorithms’ performance declines sharply. Though the current literature has studied IL in the context of imperfect demonstrations [Wu et al., 2019], the setting of mixture dataset has not been explicitly studied. Given that the optimality of the trajectories should not be assumed in practice, this shortcoming seems to have a profound impact on their usability in real world IL environments.

Motivated by these considerations, another goal of this thesis is to design a new IL algorithm that can be
trained efficiently while being robust to the quality of expert data. Building on recent advances in batch reinforcement learning (BRL) and self-supervised learning (SSL), I propose a novel batch imitation learning (BIL) algorithm, Disagreement Regularized Batch-Constrained-Q Imitation Learning (DRBIL) that satisfies these desiderata. DRBIL inherits the core architecture of a state-of-art BRL algorithm, Batch Constrained Deep Q-Learning (DRBCQ) [Fujimoto et al., 2018a], and learns a policy that optimizes for the agent’s performance after its actions are perturbed. Since DRBIL does not receive RL rewards to train its policy, it incorporates uncertainty cost [Brantley et al., 2020], a self-supervised learning technique, that learns a reward function by discriminating optimal and sub-optimal transitions in the expert dataset.

Combining two techniques that improve a policy’s robustness to sub-optimal samples, DRBIL is able to achieve state-of-art performance on the MuJoCo continuous control suite [Todorov et al., 2012] and significantly outperforms baseline methods when the expert dataset given includes degraded trajectories - while doing so in a complete offline manner, without any interaction with the environment. Compared to GAIL, its offline-learning nature makes it more practical to train, as interactive access to the environment can be risky in some real-life IL tasks. Compared to BC, which also operates in batch mode, DRBIL is robust to unreliable expert trajectories. Hence, a robust BIL algorithm like DRBIL is more likely to be deployed in practice.

The results in this thesis, collectively, call for a greater scrutiny of existing IL methods and research practices, as well as demonstrating the promise of a novel batch imitation learning framework (BRL + SSL), which DRBIL makes a first attempt. We now conclude the chapter by signposting the rest of the thesis and then reviewing related work.

In summary, the main contributions of this thesis are:

- A collection of evaluation metrics for imitation learning algorithms (Chapter 3)
- A deeper understanding of two baseline algorithms (BC & GAIL) that extends the state-of-art understanding (Chapter 4)
- A novel robust batch imitation learning algorithm DRBIL that demonstratbly learns a robust policy (Chapter 5 & 6)

This thesis is organized as follows: Chapter 2 reviews reinforcement learning and provides an overview of imitation learning, including its problem definition, baseline algorithms (BC & GAIL), and applications. Chapter 3 introduces the suite of IL evaluation metrics to be used throughout this thesis. Chapter 4 benchmarks BC and GAIL against the proposed metrics and illustrates these algorithms’ shortcomings. Chapter 5 introduces batch imitation learning and surveys a state-of-art algorithm, Batch Constrained
Q-Learning (BCQ). Chapter 6 presents DRBIL and evaluate its performance. Chapter 7 synthesizes our findings and discuss future research directions. Finally, Chapter 8 concludes.

1.1 Related Work

Our proposal of evaluation metrics and benchmarking of popular algorithms is similar to those efforts in the reinforcement learning literature. Duan et al. [2016] extensively benchmark continuous RL algorithms on the Mujoco suites. Machado et al. [2018] provide a standard guideline for evaluation in the Atari 2600 environments. Henderson et al. [2018] provide an excellent treatment of the fragile empirical foundation of deep RL and calls for a tighter standardization of experimental reporting. This thesis is, however, the first such evaluation study of its kind in the field of imitation learning.

This thesis also relates to the existing literature in batch reinforcement learning [Lange et al., 2012], self-supervised learning [Pathak, 2019], and batch imitation learning [Osa et al., 2018]. DRBIL builds on top of BCQ [Fujimoto et al., 2018a], which learns a robust policy parameterized by Q-function through action perturbation; however, BCQ is not the only modern deep BRL algorithm that has drawn much attention to this sub-field. Kumar et al. [2019] Agarwal et al. [2019] are two other works that discover the vulnerability of off-policy RL¹ and propose regularization techniques to stabilize the training of off-line Q-learning.

Self-supervised learning [Pathak, 2019] has become an emerging field in machine learning that performs supervised learning by learning suitable target signals through the training data first. It has been applied to many subfields in RL such as exploration [Pathak et al., 2019], model-based learning [Pathak et al., 2017], and more recently, imitation learning Brantley et al. [2020], which first derived the uncertainty cost function that is used in DRBIL.

Finally, DRBIL is a batch imitation learning algorithm, which has surprisingly shown little progress since Behavior Cloning [Pomerleau, 1989]. To the best of my knowledge, Kostrikov et al. [2020] is the only other published work that proposes some BIL algorithm. The main algorithm in Kostrikov et al. [2020] relies on using weighted importance sampling to train a policy gradient objective using off-line samples; the pure batch variant is only discussed in the appendix and only experimented with one simple environment. In contrast, DRBIL is modular, conceptually simple, and extensively tested in this thesis.

¹A type of RL approach that learns using samples ont collected by the current policy. See Kumar et al. [2019] for detail.
Imitation is not just the sincerest form of flattery - it’s the sincerest form of learning.

George Bernard Shaw

2

Imitation Learning Preliminaries

In this chapter, we provide a brief exposition of IL. We begin by reviewing RL and IL, including their problem formulations as well as basic solution concepts. Then, we present two baseline imitation learning algorithms: Behavior Cloning (BC) Pomerleau [1991] and Generative Adversarial Imitation Learning (GAIL) Ho and Ermon [2016]. This chapter concludes with a brief survey of use cases of IL in practice. Notations introduced in this chapter will also be used throughout the rest of this thesis.

2.1 Reinforcement Learning

In reinforcement learning (RL), an agent interacts with an environment sequentially and learns a good policy. At each time step, the agent chooses an action and receives a reward signal associated with taking the action ¹. We formalize this framework by considering a Markov Decision Process (MDP) \( \mathbb{M} \). That is, the agent interacts with a MDP \( \mathbb{M} \) defined as a tuple \((S, A, P, R, \gamma)\). \( S, A \) represent the (finite) state and action spaces; \( P(s'|s, a) \) and \( R : S \times A \to \mathbb{R} \) represent the transition distribution and the reward function, and \( \gamma \in (0, 1) \) is the discount factor. The agent starts at an initial state \( s_0 \sim P(s_0) \). At each time step \( t \), the agent samples an action \( a_t \) according to a policy \( \pi_\theta(a_t|s_t) \) parameterized by \( \theta \), and observes reward \( r_t = R(s_t, a_t) \). Then, the agent transitions to \( s_{t+1} \) according to the (Markovian) transition distribution. This generates a trajectory \( \tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T) \), and then the likelihood of \( \tau \) under

¹The action selection process and the reward need not be deterministic.
policy $\pi_\theta$ is then $p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$. The goal of reinforcement learning is to find an optimal policy that maximizes the expected discounted sum of rewards along trajectories

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] = \mathbb{E}_{\tau \sim p_\theta(\tau)}[R(\tau)]$$

That is, $\theta^* = \arg\max_{\theta} J(\theta)$. Now, we briefly review the two main approaches for solving $\theta^*$: policy gradient and Q-learning.

### 2.1.1 Policy Gradient

Policy gradient [Sutton et al., 2000] based methods [Schulman et al., 2015a] [Schulman et al., 2015b] [Schulman et al., 2017] directly optimize the RL objective $J(\theta)$ with respect to $\theta$. Using the REINFORCE trick [Williams, 1992], it can be shown that policy gradient can be estimated using $N$ trajectories from the current policy $\pi_\theta$

$$\nabla \theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=0}^{T} \nabla \theta \log \pi_\theta(a_t|s_t) \right) \left( \sum_{t=1}^{T} r(s_t, a_t) \right)$$

Then, an update to the policy parameters $\theta$ can be taken with a gradient optimizer, such as Adam [Kingma and Ba, 2014].

### 2.1.2 Q-Learning

Instead of directly optimizing for the objective, which has proven to be unstable [Schulman et al., 2015a], Q-learning [Watkins and Dayan, 1992] based methods aim to discover a policy that maximizes the action-value function, or the Q-function. For any policy $\pi$, the Q-function, denoted $Q^\pi(s, a)$, is defined as the expectation of cumulative (discounted) rewards,

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a, s_t \sim P(\cdot | s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot | s_t), r_t \sim R(s_t, a_t) \right]$$

The optimal policy $\pi^*_\theta$ attains the maximum expected return. That is, $Q^{\pi^*_\theta}(s, a) \geq Q^\pi(s, a)$ for all $\pi, s, a$. The Q-function of the optimal policy, $Q^{\pi^*}(s, a)$, can then be characterized by the Bellman optimality equation [Bellman, 1954]:

$$Q^{\pi^*}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q^\pi(s', a') | r \sim R(s, a), s' \sim P(\cdot | s, a) \right]$$
Q-learning based methods Mnih et al. [2015] Hessel et al. [2018] iteratively approximate $Q^*(s, a)$ by performing regression. In the simplest formulation [Watkins and Dayan, 1992], an agent executes its current policy and updates the Q-value of each state-action pair it visits:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

until the Q-values converge.

2.2 Imitation Learning

While RL has led some of the most canonical breakthroughs in AI in recent years [Mnih et al., 2015] [Silver et al., 2016], its success has been largely limited to simulated domains, in which a well-defined reward signal is given. While it may be intuitively clear what the reward function should be in simulated domains (e.g. game score in Atari games) [Mnih et al., 2013], in many real-world domains, the assumption of having access to a reward function is too strong. Consider the task of learning a good driving policy. While a good human driver is able to drive safely on roads, he/she may not be able to mathematically formulate a reward function that accurately separates good and bad driving policies. Without a good reward function, RL is ill-posed to solve the autonomous driving problem [Codevilla et al., 2019].

Despite the difficulty of defining a reward function, a good policy can still be learned by directly imitating trajectories provided by an expert, who may or may not have access to the true reward function, but nonetheless knows how to behave optimally in the environment. Ideally, the policy does not merely memorize the expert trajectories, but generalizes expert behavior in unseen states in the expert trajectories. This approach to learning a policy by mimicking is broadly referred to as imitation learning (IL).

It is also worthwhile mentioning that learning a good policy without reward by imitation is not the only motivation for IL. In some settings [Carroll et al., 2019] [Hu et al., 2020], such as Human-AI Coordination which we study below, the objective is purely imitating an (human) agent’s behavior in an environment, and not achieving some pre-specified (expert) performance. In these cases, the reason for using IL techniques becomes apparent.

Formally, the learning agent is provided with a set of sequential expert trajectories $D := \{ \tau_1, \ldots, \tau_N \}$ that comes from a fixed, unknown expert policy $\pi_{exp}$ (human) interacting in the task MDP $\mathcal{M}$. Each $\tau_i = (s_o, a_o, \ldots)$ denotes a single episode from $\pi_{exp}$; note that trajectories need not to be of the same
length in environments where episode durations naturally vary. We denote the total number of samples in
the expert dataset as $|D| = \sum_{i=1}^{N} |\tau_i|$.

The goal is to learn a good behavior policy $\pi$ using $D$ without access to $\pi_{\text{exp}}$. The learning is usually
achieved through minimizing a loss function that captures the notion of closeness between the imitator
policy and the expert policy. For example, Behavior Cloning uses the negative log-likelihood loss, and
GAIL uses the cross-entropy loss. The learning procedure can also differ in whether the training is
completed online or offline. For example, GAIL collects samples from the environment to improve its
policy much like RL algorithms, while BC does not collect more samples and improves its policy purely
through batch, iterative policy improvement.

Now, we examine BC and GAIL in detail, summarizing the algorithms as well as explaining their
limitations, as understood in the current literature. A goal of our evaluation effort in the next two chapters
is to verify and quantify the extent to which these limitations hold empirically. Before we dive into each
algorithm, we present a simple categorization of the two methods based on common IL algorithm
features to preview their main differences.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Batch?</th>
<th>RL?</th>
<th>Loss Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavior Cloning</td>
<td>Yes</td>
<td>No</td>
<td>Negative Log-Likelihood Loss (NLL)</td>
</tr>
<tr>
<td>GAIL</td>
<td>No</td>
<td>Yes, TRPO²</td>
<td>Cross-Entropy Loss</td>
</tr>
</tbody>
</table>

2.2.1 Supervised Learning - Behavior Cloning

A simple and popular approach to IL is to cast the problem as a standard supervised learning problem.
That is, for any given expert state-action pair $(s, a)$, we treat the state $s$ as the label and the action $a$ as the
target. Then, IL becomes a classification or regression problem (depending on if the action space is
discrete or continuous) with state as the input and action as the output. Behavior Cloning (BC) then
solves for a policy by minimizing the supervised training loss:

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

Compared to most other IL algorithms, BC is easy to implement and does not require any interaction
with the MDP. Therefore, in many settings in which interaction with the environment is risky or
expensive [Achiam et al., 2017] [Codevilla et al., 2019], BC is usually the default baseline.
2.2.1.1 Limitations of Behavior Cloning

In this section, we summarize limitations of BC that are recognized by the current literature. In Chapter 4, the benchmarking effort suggests that some of these limitations may be overstated.

Distributional shift. The main problem with BC is that its performance greatly suffers in the presence of distributional shift. In supervised learning, distributional shift describes the phenomenon of the training and testing data coming from different distributions [Sugiyama et al., 2017]. When this occurs, a classifier trained on the training set cannot guarantee good performance on the testing set, simply because any patterns it may capture from the training set may not be present in the testing set. By reducing imitation learning to supervised learning, BC inherits the distributional shift problem. Encountering a state that it has not seen in the training set, a BC imitator may perform arbitrarily bad in the current step [Ross and Bagnell, 2010].

Compounding error. Consequently, BC suffers from compounding error: a bad action now will lead to more bad actions, making the entire trajectory severely sub-optimal. Supervised learning assumes that each entry in the data set is independent and identically distributed. In IL, this assumption is clearly violated as state-action pairs from the expert trajectories are sequential and Markov, by definition. Similarly, the BC imitator’s predicted action execution also affects its future states and thereby actions. As a result, any mistake (bad action) that the imitator makes may lead the imitator to an unfamiliar state and execute more bad actions.

If we think of the imitator executing its learned policy as an online learning problem, then BC incurs a quadratic regret [Ross and Bagnell, 2010] [Ross et al., 2011]. In words, regret of a policy is the positive difference between the expert policy’s utility (cumulative rewards) and the policy’s utility. Formally, given a policy $\pi$, let $d_\pi \triangleq \frac{1}{T} \sum_{t=1}^{T} d_s$ encodes the state visitation frequency over $T$ time steps from deploying $\pi$. Let $e(s, a) = \mathbb{I}[a \sim \pi_E(s)]$, the event that action $a$ is not the action taken by the (deterministic) expert. Denote the expert policy by $\pi_E$. Then, we let $e_\pi(s) = \mathbb{E}_{a \sim \pi(s)} [e(s, a)]$ denote the expected 0-1 loss of policy $\pi$ in state $s$. Using these notations, we can capture the idea of quadratic regret precisely.

**Theorem 2.2.1.** [Ross et al., 2011]. Given an expert policy $\pi_E$, suppose a BC imitation policy $\hat{\pi}$ makes a mistake with at most probability $\epsilon$, i.e. $\mathbb{E}_{s \sim d_\pi} [e_\pi(s)] \leq \epsilon$. Then, $J(\hat{\pi}) \leq J(\pi_E) + T^2 \epsilon$, where $J(\cdot)$ is the cumulative rewards collected in $T$-steps.

To mitigate the distributional shift and compounding error problems, BC requires a large number of expert demonstrations to obtain reasonable performance [Jeon et al., 2018] [Sasaki et al., 2018]. The intuition is simple: with a large number of expert demonstrations that cover the state space well, BC can

\(^3\pi_E\) is unknown to the imitator, but given here for the sake of analysis.
learn a policy that is unlikely to encounter unfamiliar states and execute bad actions.

In summary, BC is a simple IL baseline algorithm, but it does not have the ability to reason about the sequential nature of the expert samples. This purely supervised learning approach to IL limits its ability to generalize to unseen states and actions, consequently reducing its practicality in stochastic environments. Despite these limitations, BC learns in batch mode, meaning that even it learns a policy that is not desirable for some reason, its learning process itself is not risky because it does not require interacting with the environment. This feature of BC is to sharp contrast to the interactive nature of GAIL, which we now introduce.

2.2.2 Generative Adversarial Imitation Learning

The perceived limitations of Behavior Cloning call for the need of an IL algorithm that is capable of performing long-horizon action planning. Addressing this problem, Generative Adversarial Imitation Learning (GAIL) [Ho and Ermon, 2016] has become the baseline state-of-art method that lays the algorithmic and theoretical foundations for much of the current progress in IL.

Inspired by Generative Adversarial Network (GAN) [Goodfellow et al., 2014], GAIL optimizes a mini-max game between the generator whose job is to generate trajectories that resemble the expert as closely as possible and the discriminator whose job is to distinguish between expert trajectories and generator trajectories. The discriminator $D_\phi$ is represented by a neural network with parameters $\phi$, and the generator is represented as a reinforcement learning policy $\pi_\theta$. Then, GAIL optimizes for the following min-max cross-entropy objective [Blondé and Kalousis, 2018]:

$$\min_{\theta} \max_{\phi} V(\theta, \phi) \triangleq \mathbb{E}_{\pi_\theta}[\log(1 - D_\phi(s, a))] + \mathbb{E}_{\pi_\epsilon}[\log D_\phi(s, a)] \quad (2.7)$$

In practice, the objective also includes a causal entropy [Ziebart et al., 2008] term on the policy $H(\pi_\theta)$ as a policy regularizer, encouraging the solution to not overfit. The solution alternates between taking a gradient step w.r.t $\phi$ to increase $V(\theta, \phi)$ and a RL policy optimization step (e.g. TRPO [Schulman et al., 2015b]) on $\theta$ to decrease $V(\theta, \phi)$. Intuitively, the discriminator $D$ is trained as a binary classifier that distinguishes trajectories from $\pi_\theta$ and $\pi_\epsilon$, and the generator is a RL optimization whose goal is to learn $\pi_\theta$ using the outputs of discriminator as rewards. For clarity, we present a graphical illustration for GAIL to explains its training procedure.
Figure 2.2.1: GAIL follows a 3-step procedure: 1. The imitator collects trajectories. 2. The discriminator update itself using both expert and imitator trajectories. 3. The discriminator sends signal to the imitator/generator to improve its policy via RL.

The whole algorithm is presented below as Algorithm 1 (reproduced from [Ho and Ermon, 2016]).

**Algorithm 1 Generative Adversarial Imitation Learning**

**Require:** Expert trajectories $\tau_E \sim \pi_E$

**Require:** initial policy and discriminator parameters $\theta_0$, $\phi_0$

1: for $i = 0, 1, 2, \ldots$ do
2: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
3: Update the discriminator parameters from $\phi_i$ to $\phi_{i+1}$ with the gradient

$$\hat{E}_\tau [\nabla \phi \log (1 - D_{\phi_i}(s, a))] + \hat{E}_{\tau_E} [\nabla \phi \log (D_{\phi}(s, a))] \quad (2.8)$$

4: Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log (1 - D_{\phi_{i+1}}(s, a))$. Specifically, take a KL-constrained natural gradient step with

$$E_{\tau_i}[\nabla \theta \log \pi_{\theta}(a|s)Q(s, a)] - \lambda \nabla \theta H(\pi_{\theta})$$

where $Q(\bar{s}, \bar{a}) = \hat{E}_{\tau_0} [\log (1 - D_{\phi_{i+1}}(s, a))] | s_0 = \bar{s}, a_0 = \bar{a} \quad (2.9)$

5: end for

Parallel to its intuitive appeal, GAIL’s min-max objective enjoys a nice information theory interpretation that it corresponds to minimizing the Jenson-Shannon divergence between the two policies. This results
relies on the key insight that the closeness between two RL policies can be characterized by their difference in occupancy measure, as measured by some convex function. First, we define the occupancy measure of a policy, which captures the distribution of state-action pairs that an agent encounters while deploying its policy in the targeted MDP. If running policy $\pi$ on MDP $M$ induces an ergodic Markov chain, then we can interpret the occupancy measure of a policy as the stationary distribution of the induced Markov chain.

**Definition 2.2.1.** For a policy $\pi \in \Pi$, define its occupancy measure $\rho_\pi : S \times A \rightarrow \mathbb{R}$ as

$$\rho_\pi(s, a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi).$$

Then, a nice result is that there exists a bijection between a policy and an occupancy measure.

**Proposition 2.2.1.** [Ho and Ermon, 2016]. Denote the set of valid occupancy measures $\mathcal{D} \triangleq \{ \rho_\pi : \pi \in \Pi \}$. Then, if $\rho \in \mathcal{D}$, then $\rho$ is the occupancy measure for $\pi_\rho \triangleq \rho(s, a)/\sum_{a'} \rho(s, a')$, and $\pi_\rho$ is the only policy whose occupancy measure is $\rho$.

Therefore, to induce a policy that is similar to the expert policy, we just need to match their occupancy measures under some metric. For a more detailed derivation, we refer interested readers to [Ho and Ermon, 2016].

In practice, GAIL often fails to find the globally optimal solution to the min-max objective and settles for a local optimum; this sub-optimality is inherited from the training instability of GAN and policy-gradient RL algorithms. Yet, it demonstrates excellent sample-efficiency w.r.t expert demonstrations, often able to discover good policies with as few as just one expert trajectory [Ho and Ermon, 2016]. Due to its simple algorithmic appeal as well as its elegant theoretical foundation, GAIL has become the baseline algorithm that serves as an important bedrock for imitation learning research.

### 2.2.2.1 Limitations of GAIL

Likewise, in this section, we summarize the perceived limitations of GAIL in the literature.

**Training instabilities.** An instantiation of the GAN framework, GAIL inherits many of the training instability issues that GAN suffers [Kodali et al., 2017] [Arjovsky et al., 2017]. In particular, GAIL is difficult to tune in practice. Often, a careful parameter search is required to obtain good performance. A resulting issue is that if the discriminator is too powerful (e.g. achieving high accuracy on both expert and imitator samples), then the reward signal passed down to the generator is too sparse to learn any meaningful policy [Sasaki et al., 2018].
**Interactive Access.** As GAIL iteratively interacts with the environment to improve its policy, it requires interactive access to the environment. In RL and IL, interacting with environment is usually required for any form of optimality guarantee [Sutton and Barto, 2018] [Ross and Bagnell, 2010] [Ross et al., 2011]; however, this can also turn out to be a limitation because safe interaction cannot always be assumed. First, not all environments can be directly interacted with, or direct interaction is risky. If the environments are Atari game environments [Bellemare et al., 2013], then the game itself is a virtual simulation in which interaction is straightforward and cheap. But consider the autonomous driving setting. Direct interaction means having the autonomous vehicle drive on the road itself and improves its policy iteratively. This is unfeasible and risky because doing so may damage the road, the vehicle, or even a person. Therefore, without safety guarantee built into the algorithm, the requirement of environmental interaction is a prohibitive restriction. In environments where direct interaction is costly, a simulator may circumvent such need. However, building a simulator is a difficult task on its own [Ha and Schmidhuber, 2018] [Kaiser et al., 2019]. Even in cases where simulators can be built, they are usually inaccurate [Codevilla et al., 2019] and the learned policies fail to transfer to the real environment [Tan et al., 2018].

**Sample Inefficiency.** GAIL’s RL optimization step (2.9) uses TRPO [Schulman et al., 2015b], an on-policy policy gradient algorithm. Despite its theoretical appeal, a major disadvantage of TRPO and other on-policy methods such as PPO Schulman et al. [2017] or A2C Mnih et al. [2016] is that they are sample-inefficient w.r.t to the number of interactions required with the environment; the source of inefficiency comes from the fact that to correctly estimate the current policy gradient, samples from the current policy must be obtained from interacting with the environment. In other word, to improve the policy, at every update step, new samples must be collected. As GAIL uses on-policy RL algorithm as a subroutine, it necessarily inherits this problem. Coupled with unstable reward signal from training the discriminator, the sample inefficiency of GAIL is magnified and renders it difficult to use in problems where interacting with the environment is risky or costly.

### 2.3 Imitation Learning Applications

In this section, we briefly survey key imitation learning applications. Understanding the real use cases of IL will inform us what evaluation metrics are needed in practice to ensure that IL algorithms are deployable in these applications.

#### 2.3.1 Applications

**Autonomous Driving.** Historically, the growth of imitation learning as a field hinged upon a growing interest in developing autonomous driving vehicles since the beginning of the 1990s [Pomerleau, 1989]

---

[4]Later GAIL-inspired methods, such as Kostrikov et al. [2019], use more sample-efficient RL algorithms.
Quest for fully autonomous vehicles has recently gained a new found interest with the birth of deep learning that enables modelling of much more complex navigation and road situations [Grigorescu et al., 2019] that were previously out of reach for older methods. Though possible to learn through reinforcement learning [Vitelli and Nayebi, 2016], good driving behavior is not easily captured by a reward function. Hence, learning from real human driving demonstration has become the predominant method for learning an autonomous driving policy [Codevilla et al., 2019] [Zhang and Cho, 2016] [Pan et al., 2017]. Despite the abundance of human driving data, learning a driving policy directly in the real world is risky as unstable learning may cause deterioration of the road or the vehicle itself. As such, policy learning is usually achieved through imitation learning in a driving simulation environment, such as the popular CARLA driving simulator [Dosovitskiy et al., 2017], and then transferred to a real driving environment.

**Robotics Manipulation.** Another popular imitation learning application is learning dexterous robot: training a robot arm to achieve some object manipulation task. Below, we show a static illustration of training a robot arm to learn to push an object to the desired location. In contrast to autonomous driving, defining a reward function is tractable [Yu et al., 2019]. For sparse reward, a positive reward can be
assigned to an episode that successfully achieves the desired manipulation task. For dense reward, a reward proportional to the degree that the desired task is achieved may be assigned. Imitation learning is preferred to reinforcement learning in these problems because exploration is difficult and almost impossible, while imitating an expert demonstration is simple [Akkaya et al., 2019]. Notably, imitation learning in these settings is efficient and can be often achieved in one-shot fashion [Pathak et al., 2018] [Finn et al., 2017] [Duan et al., 2017].

**Human-AI Coordination.** Systems involving an human and AI (robot) to cooperative has increasingly become an important application of multi-agent system [Dragan et al., 2015] [Bauer et al., 2008]. In these settings, training the cooperative AI usually involves it acquiring an internal model of human behavior. This human modelling is usually achieved through pure algorithmic IL [Carroll et al., 2019] or a combination of machine learning and cognitive behavioral model [Sadigh et al., 2016]. Since the goal of AI in these settings is to assist human, learning an accurate human behavior model through imitation learning is often the key that determines the success of such systems.

### 2.4 Discussion

In this chapter, we review basic concepts and notations in RL and IL. Additionally, we briefly surveyed BC and GAIL, two baseline imitation learning algorithms. Finally, we presented some important real-world IL applications that illuminate desired properties of imitation learning algorithms in these settings, which helps guide our presentation in the following chapters. In the next chapter, we dive into the evaluation of imitation learning algorithms, noting the insufficiency of current approaches and making concrete suggestions to deepen our understanding of IL.
3

On Evaluating Imitation Learning Algorithms

In this chapter, we fill the gap in the current literature by proposing six metrics that evaluate IL algorithms along three axes: performance, imitation, and robustness. We begin by first surveying the evaluation metrics used in existing works; this portion of the chapter can be read as a mini literature review. We argue that these metrics are incomplete and fail to capture key attributes that an IL algorithm should fulfill in practice. Then, we present the suite of new evaluation metrics, which we will use throughout the rest of this thesis to evaluate IL algorithms.

3.1 Evaluation Metrics in Existing Work

In this section, we survey existing evaluation metrics in the literature. We conclude that they overwhelmingly emphasize the performance of an IL algorithm and ignore critical aspects of imitation and robustness.

3.1.1 On Evaluating Performance

The current literature evaluates new algorithms primarily by their ability to match the expert’s performance (cumulative rewards) in the environment. This is done by comparing the imitator’s performance after training is complete to that of the expert. The imitator’s performance is approximated by collecting trajectories using the imitator’s policy in the environment and computing the average and
the standard deviation of the cumulative rewards of the trajectories collected. The expert’s performance is approximated by the average of the cumulative rewards of the expert trajectories given to the imitator. This metric is widely used in existing work and serves to measure the sample complexity of IL algorithm with respect to the expert trajectories. For example, the original GAIL paper [Ho and Ermon, 2016] demonstrates the superiority of GAIL to prior methods by showing that GAIL is able to achieve performance close to that of the expert with very few expert trajectories. For concreteness, we reproduce the results shown in Ho and Ermon [2016] below.

![Performance of learned policies shown in the original GAIL paper](image)

**Figure 3.1.1:** Performance of learned policies shown in the original GAIL paper [Ho and Ermon, 2016].

This metric is indeed useful in that it identifies IL algorithms suitable for regimes where collecting expert data is difficult. However, this metric alone is far from constituting a thorough understanding of the desirability of an IL algorithm. First, it does not inform the sample efficiency of the algorithm with respect to the number of interactions in the environment. Methods such as GAIL require interacting with the environment to receive feedback. An algorithm that takes an exorbitant number of environmental samples to achieve expert performance is no different from another algorithm that is sample efficient under this metric. As such, in environments in which interacting with the environment is expensive (e.g. autonomous driving), this metric is not a good discriminator for good IL algorithms.
Similarly, this metric only considers the performance of the final, learned imitator policy. Exhibiting high variance during training is an undesired property that may have immense negative consequence in risky environments, yet this problem is hidden and unaddressed if we solely rely on this metric. Existing works that do report the entire training curve [Sasaki et al., 2018] [Blondé and Kalousis, 2018] only report the performance average, hiding the possibility that each individual run exhibits extreme performance fluctuation during training - which we find to be indeed the case in our experiments.

3.1.2 On Evaluating Imitation

The current literature primarily evaluates an imitation learning algorithm by its ability to achieve similar performance to the expert, if not better, and disregards its ability to imitate.

Certainly, in some applications, achieving expert or even better performance is a sensible goal, and much of existing work optimizes imitation learning algorithm with performance as the ultimate goal. But performance evaluation is dependent on the environment having a reward function, using which performance can be well defined (e.g. cumulative rewards). However, in some settings, such as autonomous driving, reward function cannot be assumed to be a part of the environment. Further, in many other settings like Human-AI Collaboration, matching the expert behavior is directly the goal of IL.

It is then only possible and necessary to evaluate the algorithm by its ability to match the expert as closely as possible in their behaviors. That is, the imitator and the expert should take similar, if not identical\(^1\), sequence of actions starting at the same state. Note that if this condition is achieved, then their performances in the environment will naturally be matched, given that the reward is reasonably deterministic (i.e. the distribution of \(r(s, a)\) has low variance). The converse is not true in most cases, however. Two different policies that are qualitatively different can achieve the same performance. For example, consider the TREE-PASSING task illustrated below. Policy LEFT and RIGHT achieve the task of passing the tree equally well, yet they clearly are two different policies. Without evaluating IL algorithms on their faithfulness to the expert policy, we risk undesirable behaviors from the algorithms.

Even in environments that assume reward function and hence performance-based evaluation, it is useful to evaluate algorithms based on their ability to imitate. Notably, algorithms such as GAIL are motivated from a theoretically grounded framework of occupancy matching, yet whether the algorithm actually adheres to the theory well is unknown because the original paper only reports the results shown above, which is clearly not sufficient for evaluating imitation. To the best of our knowledge, no subsequent work of GAIL [Li et al., 2017] [Peng et al., 2018] [Blondé and Kalousis, 2018] has investigated or addressed

\(^{1}\)note that this is not possible in environments with continuous action space
this issue in depth. GAIL and its variants are all given green light in the literature because their ability to achieve good final performance.

For recent works that do compare the imitator and the expert’s behavioral difference, their studies are preliminary and limiting, at best. For example, Santara et al. [2018] finds that the GAIL’s distribution of trajectory-rewards is more heavy-tailed than the expert, but does not study the distributional difference in trajectories themselves. In IL applications in which imitation is the sole goal, such as Human-AI coordination, the model trained using IL is treated as a black-box model of human behavior without a rigorous attempt at evaluating the quality of imitation Carroll et al. [2019].

Additionally, faithful imitation is often necessary in real use cases. In Human-AI interaction settings, the goal of imitation learning is to model the behavior of an (human) agent in a multi-agent system. This learned human model in turn is used to help the AI learn cooperative behavior that complements that of the human. If the human model, learned through imitation learning, does not behave like a human, then the learned AI model will exhibit sub-optimal behavior when paired with a real human ². This phenomenon is widely documented [Carroll et al., 2019] [Dimitrakakis et al., 2017] and carries with heavy consequences in domains where the adoption of AI is meant to complement and cooperative with human. To ensure the safety and deployability of real AI system, it is crucial that the imitation capability of IL algorithms studied in depth.

3.1.3 ON EVALUATING ROBUSTNESS

In addition to insufficient measures of studying IL algorithms’ ability to perform and imitate, their robustness is also understudied. This lack of attention is partially due to the wide adoption of iterative imitation learning algorithms in practice [Ross and Bagnell, 2010] [Ross et al., 2011]. In iterative settings,

²This is equivalent to the distributional shift problem we studied before.
the imitator is allowed to iteratively ask the expert for feedback (e.g. label) or more data while it is learning a policy. This iterative process enables the imitator to continuously improve its policy and mitigate distributional shift and compounding error using expert feedback.

However, it is clear that expert reinforcement is not readily available in all IL tasks. Hence, robustness is an intrinsically desirable property that deserves proper attention. Laskey et al. [2017] study the issue of distributional shift but suggests training algorithms with noise injection, and does not propose metrics that evaluate performance of algorithms in settings with unexpected stochasticity. Reddy et al. [2019] propose an evaluation metric in which the initial state $s_0$ during the testing phase is different from that of the training phase. This method fails to account for distributional mismatch and stochasticity that may arise during the middle of an episode, and hence is not a convincing measure of robustness.

3.2 A Set of Imitation Learning Evaluation Metrics

We suggest that a complete suite of evaluation metrics for imitation learning needs to study the following three different aspects of an IL algorithm: performance, imitation, robustness. We include metrics from the literature as well as introducing new ones that study questions that were not addressed by existing works.

Our evaluation includes six distinct metrics, two along each axis mentioned. Since many metrics are qualitative and best described in words, we begin the section with a summary table that captures the essential information about the metrics. Then, we devote the rest of the chapter describing each of them in detail.
### Table 3.2.1: Imitation Learning Evaluation Metrics

<table>
<thead>
<tr>
<th>Method</th>
<th>Short</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Performance</td>
<td>BP</td>
<td>Performance</td>
<td>Compute and compare the best performance achieved during training across different sizes of expert data.</td>
</tr>
<tr>
<td>Running Performance</td>
<td>RP</td>
<td>Performance</td>
<td>Report the model performance during training from iteration to iteration (e.g. training curve).</td>
</tr>
<tr>
<td>Stepwise Likelihood</td>
<td>SL</td>
<td>Imitation</td>
<td>Compute and plot the stepwise likelihood of expert trajectories under $\pi_{imitator}$.</td>
</tr>
<tr>
<td>Cumulative-Statistics Histogram</td>
<td>CH</td>
<td>Imitation</td>
<td>Collect trajectories from $\pi_{imitator}$ and construct histograms of their cumulative rewards and durations. Compare with those of expert trajectories.</td>
</tr>
<tr>
<td>Noisy Actions</td>
<td>NA</td>
<td>Robustness</td>
<td>Inject randomness to the imitator action outputs. Compare the model’s performance with that of an un-perturbed model.</td>
</tr>
<tr>
<td>Unreliable Expert</td>
<td>UE</td>
<td>Robustness</td>
<td>Train imitator models using expert data that contains sub-optimal transitions. Compare its results on the two performance metrics with that of a pure-expert imitator model.</td>
</tr>
</tbody>
</table>

#### 3.2.1 Performance

#### 3.2.1.1 Best Performance

We first would like to understand the best possible performance an IL algorithm can achieve during its training. Clearly, we should not deploy an algorithm that is unable to achieve expert performance \(^3\) during any point of its training. Therefore, we suggest the following:

*Best Performance*

Plot the best cumulative rewards achieved during training vs. number of expert trajectories (Figure 3.1.1).

This metric answers two important questions we would like to know about an IL algorithm: (1) Can the algorithm achieve expert performance? (2) How many expert trajectories does the algorithm need to be trained with to do so?

\(^3\)we acknowledge that it is difficult to speak with certainty if an algorithm can "never" achieve expert performance.
3.2.1.2 Running Performance

In addition to the best performance attained, the algorithm’s characteristics during training also need to be understood. An algorithm that exhibits exorbitantly high fluctuation in performance from episode to episode or requires a large number of on-policy samples is not suitable to be trained in real-world domains. To gain a better understanding of an algorithm during its training phase, we suggest the following:

**Running Performance**

Plot the cumulative rewards vs. number of on-policy samples. Include the average plot over multiple seeds as well as the individual runs’ plots.

This metric answers two additional important questions regarding an IL algorithm’s performance: (1) How many samples of environmental interactions does the algorithm need to obtain expert performance? (2) Is the algorithm stable during its training?

3.2.2 Imitation

As highlighted by the TREE-PASSING example, an algorithm’s ability to imitate the expert closely needs also be directly evaluated. The rationale behind our metrics is that we should measure both the algorithm’s ability to imitate trajectories and the distribution of trajectories. Clearly, an IL algorithm must learn to imitate the expert trajectories, but this criterion in itself is not sufficient because a brute force algorithm that simply memorizes the given expert trajectories will suffice and this solution can behave arbitrarily bad out of samples. Therefore, we should also consider some evaluation metrics that study the distribution of trajectories. In practice, we do not have the oracle expert policy from which to draw samples (trajectories). As such, a validation set of expert trajectories should be withheld from the entire expert dataset to approximate the distribution of expert trajectories, as if more rollouts from the expert are collected.

3.2.2.1 Trajectory Stepwise Likelihood

The imitator is usually parameterized by a deep neural network that outputs a Gaussian distribution over action given the state. That is, \( \pi_{imitator}(\cdot | s) \sim \mathcal{N}(\mu_s, \sigma_s^2) \). The variance is fixed and constant over all states, and the mean is the parameter to be learned in imitation learning. Since, the imitator is specified by a distribution, we can directly compute the likelihood of an expert trajectory under the imitator policy.

\(^4\text{It is worth exploring if learning this parameter can lead to better performance.}\)
Given an expert trajectory $\tau_{\text{expert}} = \{s_1, a_1, \ldots, s_T, a_T\}$, its likelihood is then

$$L(\tau_{\text{expert}}) = \prod_{t=1}^{T} \pi_{\text{imitator}}(a_t|s_t)$$

(3.1)

This likelihood itself is not informative; it is the product over the entire trajectory. Because likelihood is multiplicative, even one transition that has low likelihood under the imitator policy can make the entire trajectory unlikely. Instead, it is much more informative if we reason about the likelihood of each individual transition, $\pi_{\text{imitator}}(a_t|s_t)$, relative to other transitions in the same trajectory. This simple extraction allows us to understand qualitatively which portion of an expert trajectory the algorithm imitates well (high likelihood), and which portion it does not. Hence, we propose the following:

**Trajectory Stepwise Likelihood**

For an expert trajectory $\tau_{\text{expert}} = \{s_1, a_1, \ldots, s_T, a_T\}$, plot $\pi_{\text{imitator}}(a_t|s_t)$ as a function of $t$. When there are multiple expert trajectories, plot the average stepwise likelihood as a function of $t$.

Furthermore, the stepwise likelihood is computed and graphed for both expert trajectories used to train the models and the expert trajectories collected but not used. Comparing them allows us to detect issues such as overfitting and qualitatively which parts of expert trajectories the models learn to adequately generalize.

3.2.2.2 **Cumulative-Statistics Histogram**

Imitation at the trajectory level may be difficult and certainly impossible for tasks with continuous state and action spaces (all the Mujoco tasks). As discussed, the likelihood method above is only useful in so far as informing us the relative goodness of fit for the individual step transition in the expert trajectories.

We can gain deeper insights into the imitator by considering the distributional difference of trajectories between the expert and the imitator. To this end, we suggest the following:

**Cumulative-Statistics Histogram**

Comparing the empirical distributions (histograms) of imitator and expert trajectory performance and duration by generating trajectories using both policies under the same environmental conditions (same source randomness).

These histograms of the empirical distributions are to be constructed on the same plot, allowing a graphical comparison between the policies to be made. When the environments are discrete, then the
histograms for more refined metrics such as state-conditioned action frequency, bigram frequency can also be generated.

3.2.3 Robustness

Much of recent progress in IL builds upon that of RL. GAIL improves its policy by using a model-free policy gradient RL algorithm (e.g. TRPO), and much of the subsequent effort in improving GAIL leverages RL techniques originally introduced to improve RL algorithms [Peng et al., 2018] [Kostrikov et al., 2019]. Other non-adversarial state-of-art methods [Reddy et al., 2019] [Brantley et al., 2020] also implement a RL policy improvement subroutine in their algorithms.

Given the instability and difficulty of training RL algorithms, reinforcement learning has proven to be extremely brittle and unstable in simulation benchmarks as well as real-world domains [Li, 2017]. Given IL's dependency on RL and its applications in real-world domains, it has become ever more critical to assess IL algorithms' robustness to various sources of uncertainty. In addition to environmental stochasticity, we identify sub-optimal expert demonstrations to be another source of uncertainty that may deteriorate an imitation learning algorithm's performance. Therefore, in this section, we propose two evaluation metrics that measure an algorithm's robustness to environmental and expert uncertainty, and refer to them as noisy actions and unreliable expert, respectively.

3.2.3.1 Noisy Actions

Imitation learning algorithms obtain expert performance by performing actions similar to those of the expert in states seen in the expert data. But it is unlikely that the imitator will only encounter states that are included in expert data for various reasons. First, the environment is usually stochastic, and its dynamic is unknown to the imitator. Second, in many practical applications of IL, the imitator learns its policy in a simulated environment, and then transfers the policy to the real environment. But the two environments' dynamics often mismatch [Tan et al., 2018], making the learned policy susceptible to entering parts of the state space that it has not encountered in training.

For these reasons, it is critical that we analyze an IL algorithm's robustness to uncertainty arising from the environment. To this end, we propose the following robustness test.
Noisy Actions

Given a learned imitation policy $\pi$, we evaluate its performance by collecting rollouts from the environment with the following possible ways of injecting distributional shift in states and actions:

1. At each time step, with probability $\epsilon$, the imitator takes a random action from the action space
2. At each time step, a random noise $z \sim \mathcal{N}(0, \sigma^2)$ is added to the imitator action
3. The combination of the above

That is, the new action distribution given a state $s$ is

$$\pi_{\text{noisy}}(\cdot | s) \sim \epsilon U + (1 - \epsilon)\left(\pi(\cdot | s) + Z\right)$$

Qualitatively, the first disturbance aims to test the imitator's ability to recover from an accidental mistake that is common in many real physical systems, whereas the second procedure stress tests an imitator's ability to infer plausible actions at unfamiliar states. The proposed tests are more comprehensible than the one proposed in Reddy et al. [2019], where only the initial state $s_0$ is altered.

3.2.3.2 Unreliable Expert

Another important consideration for robustness is the imitator ability’s to mimic the expert when the expert trajectories are not always optimal. Previous works have studied the cases of sub-optimal demonstrator [Wu et al., 2019], or when the expert trajectories are collected from a different task from the one the imitator is subject to [Gangwani and Peng, 2020]. To make the scope of our evaluation as broad as possible, we study the general case when the expert is unreliable at times. That is, while most of the expert trajectories are optimal, there are some degraded expert trajectories.

While previously unexplored in the literature, this setting is in fact quite ubiquitous. In the real-world, the expert demonstrations usually come from humans, who may accidentally perform a sub-optimal routine due to fatigue or distraction. Even in simulated environments, the high variance nature of a RL trained expert [Li, 2017] means that some trajectories are optimal, and some are not. While it is certainly possible to discard degraded trajectories manually, an algorithm that is robustness to such unreliable, degraded demonstrations is certainly more desired in practice, as manual inspection of trajectory quality may be tedious and expensive. To this end, we propose the following procedure:
Unreliable Expert

Given an expert dataset \( D := \{ \tau_1, \ldots, \tau_n \} \) in which all trajectories obtain expert performance, we construct an unreliable expert dataset \( D_{\text{mixed}} := D \cup U \), where \( U = \{ \tau'_1, \ldots, \tau'_k \} \) is a set of degraded, sub-optimal demonstrations from the expert, collected using the noisy version of its policy

\[
\pi_{\text{noisy\_expert}}(\cdot|s) \sim \epsilon U + (1 - \epsilon)(\pi_{\text{expert}}(\cdot|s) + Z)
\]

using the same mechanisms of disturbance as introduced in Noisy Actions (NA) test. We augment \( D \) with additional optimal expert samples \( D_{\text{add}} := \{ s_0, a_0, s_1, a_1, \ldots \} \) and denote the augmented expert dataset as \( D_{\text{good}} := D \cup D_{\text{add}} \) such that

\[
|D_{\text{good}}| = |D_{\text{mixed}}|
\]

That is, the good expert dataset and the mixed expert dataset are to have the same number of expert samples. This step is done to ensure that evaluation results are not skewed due to different sizes of good vs. mixed expert trajectories.

Using \( D_{\text{good}} \) and \( D_{\text{mixed}} \), we train two IL agents using the same algorithm and set of parameters, respectively. Finally, we compare the performance (best performance, running performance) of the two resulting agents \( \pi_{\text{good}} \) and \( \pi_{\text{mixed}} \).

### 3.3 Discussion

In this chapter, we review existing evaluation metrics in the literature and argue that they are not sufficient for a thorough understanding of an imitation learning algorithm. Filling the gap in the literature, we present a suite of evaluation metrics in three main categories of interest: performance, imitation, and robustness. In each category, we propose a few metrics that test different properties of IL algorithms. In the next chapter, we put BC and GAIL under this set of tests and present new results that deepen our understanding of these algorithms. Our findings then motivate our study of the new batch imitation learning framework and approach introduced in succeeding chapters.
In this chapter, we test BC and GAIL against our set of evaluation metrics proposed in the last chapter. Our goal is to show a proof-of-concept and shed light on some new findings of BC and GAIL. We begin by describing the general experimental setups used throughout the thesis, and then present results. We conclude with a discussion on the algorithms as well as the efficacy of our approach. A bird-eye view of our results can be found in section 4.3.

4.1 General Experimental Setups

We use the Mujoco \cite{Todorov et al., 2012} environments\footnote{See Appendix A for more details.} to conduct all experiments in this thesis. We focus on the Walker2d-v2 environment. In this environment, the policy aims to keep a stick walker “walking” for as long as possible in the limited episode duration without falling over, which prematurely terminates the episode. An illustration of A Walker2d agent completing the walking motion is provided below.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{walker2d.png}
\caption{A Walker2d agent completing the walking motion.}
\end{figure}
Figure 4.1.1: A Walker2d Agent Walking.

In real-world domains, the expert is usually a human demonstrator who performs the optimal sequence of actions manually. Because MuJoCo environments are physics simulations, we train an “expert” using a state-of-art RL algorithm, PPO [Schulman et al., 2017], which is known to perform well in MuJoCo environments. The expert policy is denoted as $\pi_E$. See Appendix B for more detail on the expert training procedure.

We collect 100 trajectories (episodes) of $\pi_E$ from the environment. The first 50 trajectories are collected naturally without noise, and the latter 50 are collected with the following two sources of uncertainty injected to the action:

1. with 0.3 probability, a random action is taken,
2. else a noise drawn from $\mathcal{N}(0, 0.3)$ is added to the action
That is, the new action distribution given a state \( s \) is

\[
\pi_{\text{noisy}}(\cdot | s) \sim \epsilon U + (1 - \epsilon)\left( \pi(\cdot | s) + Z \right)
\]

We denote the first 50 trajectories as \( D_{\text{good}} := \{\tau_1, \ldots, \tau_{50}\} \) and the last 50 trajectories as \( D_{\text{noisy}} := \{\tau_{51}, \ldots, \tau_{100}\} \).

The two disjoint sets of optimal and noise trajectories allow us to construct mixed datasets for the unreliable expert evaluation. More specifically, for each pair of algorithm and task, we train six different models, with configurations parameterized by their corresponding expert data input.

The first three are the models trained with one, three, five trajectories\(^2\) drawn from \( D_{\text{good}} \). We denote these subsets of \( D_{\text{good}} \) as \( D^1_{\text{good}}, D^3_{\text{good}}, D^5_{\text{good}} \) and their corresponding models as MODEL\(^i\)\(_{\text{good}}, \) MODEL\(^3\)\(_{\text{good}}, \) MODEL\(^5\)\(_{\text{good}} \) for \( \text{MODEL} = \text{BC, GAIL} \), respectively. Using the procedure described in chapter 3, we construct expert dataset of mixed quality \( D^i_{\text{mixed}}, D^3_{\text{mixed}}, D^5_{\text{mixed}} \) and train their corresponding models MODEL\(^i\)\(_{\text{mixed}}, \) MODEL\(^3\)\(_{\text{mixed}}, \) MODEL\(^5\)\(_{\text{mixed}} \) for each algorithm. The relationship between each dataset and its trained models is encapsulated in the following table.

<table>
<thead>
<tr>
<th>Expert Dataset</th>
<th># of ( D_{\text{good}} ) Trajs</th>
<th># of ( D_{\text{noisy}} ) Trajs</th>
<th>Trained Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D^1_{\text{mixed}} )</td>
<td>1</td>
<td>1</td>
<td>BC(^i)<em>{\text{mixed}}, GAIL(^i)</em>{\text{mixed}}</td>
</tr>
<tr>
<td>( D^3_{\text{mixed}} )</td>
<td>3</td>
<td>3</td>
<td>BC(^3)<em>{\text{mixed}}, GAIL(^3)</em>{\text{mixed}}</td>
</tr>
<tr>
<td>( D^5_{\text{mixed}} )</td>
<td>5</td>
<td>5</td>
<td>BC(^5)<em>{\text{mixed}}, GAIL(^5)</em>{\text{mixed}}</td>
</tr>
<tr>
<td>( D^1_{\text{good}} )</td>
<td>1 + ( \epsilon_1 )</td>
<td>0</td>
<td>BC(^1)<em>{\text{good}}, GAIL(^1)</em>{\text{good}}</td>
</tr>
<tr>
<td>( D^3_{\text{good}} )</td>
<td>3 + ( \epsilon_3 )</td>
<td>0</td>
<td>BC(^3)<em>{\text{good}}, GAIL(^3)</em>{\text{good}}</td>
</tr>
<tr>
<td>( D^5_{\text{good}} )</td>
<td>5 + ( \epsilon_5 )</td>
<td>0</td>
<td>BC(^5)<em>{\text{good}}, GAIL(^5)</em>{\text{good}}</td>
</tr>
</tbody>
</table>

**Table 4.1.1:** Expert Trajectories and Their Trained Models

For each model configuration, we then train five model instantiations using five distinct random seeds and the same expert trajectories. For BC, each individual model is trained for 10000 iterations using the full expert dataset. For GAIL, each individual model is trained for one million environmental steps, with PPO policy update every \( 2048 \) steps. For detailed model specifications, refer to Appendix C.

\(^{2}\)For consistency, the first one, three, five trajectories in the order that they were collected
4.2 Benchmark Results

We train and evaluate BC and GAIL on the Walker2d environment using the set of metrics proposed. For each plot and table, mean and half standard deviation (shaded band in plots) over the 5 random seeds are reported. Unless otherwise specified, we assume the underlying model used to generate the results to be MODEL$_{\text{good}}$ for each algorithm because this configuration is supposed to be the best configuration as it is trained with the most amount of expert quality data.

4.2.1 Best Performance

For each model configuration, we compute the best cumulative rewards achieved during any training iteration and compare the values across the number of expert trajectories used to train the model.

![BC Walker2d-v2 Best Performance](image)

**Figure 4.2.1:** BC & GAIL Best Performance on Walker2d-v2

³See Appendix D for results on other Mujoco environments for BC. Obtaining full results for GAIL was too computationally expensive for the scope of this work.
As shown, BC_{good} is able to achieve the expert performance, while GAIL_{good} achieves much lower performance. Additionally, BC_{good} is not sensitive to the size of expert dataset, as it achieves expert performance in all three different sizes, while GAIL_{good} exhibits improvement as the size increases. Furthermore, BC models exhibit much lower variance than their GAIL counterparts. However, to GAIL's advantage, its performance degradation from training with $D_{mixed}$ is much less severe than that of BC. GAIL_{mixed}'s best performance trails that of BC_{mixed} when trained with only one mixed trajectory, but its performance quickly exceeds BC_{mixed}'s performance as the number of trajectories increases.

That GAIL outperforms BC with mixed trajectories is consistent with the current literature's understanding of BC and GAIL. Since BC supervise-learns state-to-action mappings, it learns to mimic the degraded trajectories as well as the expert trajectories. And when more degraded trajectories are present in training when the size of the dataset increases, their presence inhibits BC from learning the expert policy because BC necessarily fits to their sub-optimality. In contrast, GAIL holds better generalization and learns a reinforcement learning policy that accounts for the sequential nature of the learned task.

While the relative favorability of GAIL to BC in the setting of mixed trajectories makes sense given the literature, the finding that their performances decline sharply is in itself novel. While previous works have studied IL when all of the data is sub-optimal [Wu et al. 2019], our setting is more general and shows that even if only a minority of data is degraded ⁴, IL algorithms still exhibit large performance sensitivity.

In real-world domains, it is also likely that some portions of the expert data are degraded for one reason or another. Therefore, solutions to this problem need to draw greater attention. And in Chapter 6, we propose one such algorithm that is capable of robust learning in the presence of degraded trajectories.

⁴Because degraded trajectories are shorter. See Appendix B
4.2.2 Running Performance

The results above suggest that BC outperforms GAIL except for its sensitivity to the expert data quality. Further, the algorithms’ running performances are compared to gain deeper insights into the models’ training procedure. BC models are evaluated every 100 iterations \(^5\); GAIL is evaluated every time the policy is updated (every 2048 environmental steps) \(^6\).

![BC Walker2d-v2 Running Performance](image1)

(a) BC

![GAIL Walker2d-v2 Running Performance](image2)

(b) GAIL

Figure 4.2.2: BC & GAIL Running Performance on Walker2d-v2

These results suggest that GAIL is able to learn and continuously improve its policy when trained with either \(D_{\text{expert}}\) or \(D_{\text{mixed}}\). However, consistent with previous works Kostrikov et al. [2019] Blondé and Kalousis [2018], they also suggest that one million samples are not enough for GAIL to converge to the expert policy, though the PPO experts are trained using the sample number of samples (see Appendix B for more detail).

In addition to its sample inefficiency, we note that the training exhibits large fluctuation - even if near expert performance is achieved in one training episode, the model can easily forget what it has learned.

\(^5\)corresponding to every 10000 expert samples
\(^6\)Therefore, its graph looks more "dense".
and drops to a much worse policy. This explains why we observe near-expert performance in the best performance plot in the preceding section (figure 4.2.1), but the results shown here are relatively much worse. This is directly illustrated by “decomposing” the average running performance plot above into the running performance for each individual seed run separately. In the following graphs, we show the training curve of three separate seeds for both GAIL\textsubscript{good} (left) and GAIL\textsubscript{mixed} (right).

Figure 4.2.3: Comparing individual runs of GAIL models in Walker2d.

These graphs indeed confirm our qualm that each individual model exhibits larger instability than what we might expect from just reading the average plot alone. Notably, the models trained with mixed expert dataset exhibit qualitative different training instability than their corresponding models trained with just expert data. This finding is not apparent at all from the average plot (figure 4.2.2) above, supporting the
importance of including these individual plots in reporting the performance of IL algorithms. For GAIL, our conclusion is that its sensitivity to poor expert samples is more serious than previous understood in the literature [Reddy et al., 2019] [Wu et al., 2019]. In fact, these plots suggest that GAIL_{mixed} models suffer from the catastrophic forgetting problem in deep neural networks [Kirkpatrick et al., 2017] [Kemker et al., 2018]: Despite what the model has learned, it quickly forgets and plunges back to (random) sub-optimal behavior.

In contrast, BC quickly achieves expert performance during training, but its performance gradually declines. The most plausible explanation is that the model is overfitting to the expert trajectories and therefore is increasingly susceptible to compounding error as training proceeds. This suggests a good BC model can be obtained by training using an order of magnitude smaller number of iterations. In the case of BC_{mixed}, the model is indeed overfitting as its running performance sharply declines from expert level to sub-optimal in the early phase of training.

In summary, our performance-based evaluations are efficacious. They illustrate the main issues facing the models clearly and convincingly. In particular, our analysis casts significant doubt to the consensus in the literature that GAIL is more preferable to BC. Not only the latter trains much faster, it also exhibits much stable training. Its only downside that we have shown so far is its quick degradation when the expert trajectories are of mixed quality. Though GAIL appears to be more robust in this regard, its sample inefficiency and instable training represent significant barrier in practice.

In the next set of imitation-based evaluations, we aim to gain a deeper understanding into both algorithms by dissecting their abilities to imitate intra- and inter-trajectories.

4.2.3 Stepwise Likelihood

We first show the average stepwise likelihood over the five expert trajectories used to train the models. Then, we also compute and compare to the average stepwise likelihood for another five expert trajectories unseen during training.
Figure 4.2.4: Average Stepwise Likelihood over 5 Walker2d Expert Trajectories

Graph (a) suggests that BC computes a similar log likelihood value for all transitions in the trained expert trajectories, but the likelihoods for unseen expert trajectories (orange in graph (b)) are strictly lower and exhibit catastrophic decline at times (from step $\approx 150 - \approx 380$). This indeed consolidates our viewpoint that BC is overfitting to the expert trajectories and training can be halted for much earlier.

In contrast, GAIL exhibits some bizarre pattern. It is able to imitate samples early in the trajectories much better than those that appear later in the trajectory. Its stepwise likelihood for later steps in the expert trajectories exhibits a cyclic pattern that mirrors the cyclic pattern of the walking motion that an expert Walker2d agent performs. This suggests that GAIL only learns parts of this walking motion well (comparatively high likelihood), which appears to last about 50 steps, and fails to mimic other parts to complete the motion. Why might this occur?

We hypothesize that it is due to the **mode collapse** [Srivastava et al., 2017] phenomenon that is ubiquitous in GAN training. Mode collapse is a GAN training failure phenomenon in which the generator only outputs samples from a few modes in the training data distribution and ignores others. Recall that GAIL implicitly implements a GAN. Therefore, it learns a local optimum where only parts of the expert trajectory yield high reward. Once it learns the local optimum, it only slowly exits, if ever. Since the graph is averaged over 5 seeds, we can reasonably conclude that this phenomenon is general, and partially explains the training instability as well as inefficiency that were observed in the
performance-based evaluations.

### 4.2.4 Cumulative-Statistics Histogram

In this test, 50 trajectories using the same environmental stochasticity are collected from both the PPO expert and the imitator models. The distributions of their cumulative rewards and durations are plotted as histograms in order to assess the inter-trajectory imitation ability of BC and GAIL.

![Histograms of Cumulative Rewards and Durations of Expert and Imitator Trajectories in Walker2d-v2.](image)

**Figure 4.2.5:** Histograms of Cumulative Rewards and Durations of Expert and Imitator Trajectories in Walker2d-v2.

These histograms suggest that the expert policy is much more concentrated towards the optimal end (right) than either BC or GAIL. In particular, even though BC is able to achieve expert level performance, its inter-trajectory distribution qualitatively differs from that of the PPO expert.
4.2.5 **Noisy Actions**

In the performance-based metrics, both BC and GAIL exhibit sensitivity to disturbance in the expert dataset. In Noisy Actions, BC\(_{\text{good}}\) and GAIL\(_{\text{good}}\) are evaluated with systematic noises added to actions degrade performance. See Chapter 3 for detail.

![Figure 4.2.6: Walker2d Noisy Actions (NA) Evaluations](image)

Consistent with our finding that BC is more sensitive to disturbance, BC again displays a greater level to performance degradation when a Gaussian white noise is introduced to its action outputs (see left column). If the action selection mechanism itself exhibits stochasticity (ε > 0), then the performance of both BC and GAIL significantly suffers, and in fact drop to a random level. This is not surprising given that a Walker2d agent needs to perform a long sequence of correct actions to successfully "walk", and a random action over the entire continuous action space makes a completed walking motion certainly impossible.

### 4.2.6 Unreliable Expert

BC\(_{\text{mixed}}\) and GAIL\(_{\text{mixed}}\) are trained (see section 4.1 for detail). Their performance results are reported in the performance-based metrics above and will not be repeated here.
4.3 Summary of Results

We summarize our findings in the following table and make the case for BC over GAIL as the baseline method to be improved upon. The table lists seven desirable properties on the left (their corresponding test’s acronym in the parenthesis) and whether BC or GAIL satisfies each one.

<table>
<thead>
<tr>
<th>Property</th>
<th>BC</th>
<th>Comment</th>
<th>GAIL</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Performance (BP)</td>
<td>✓</td>
<td>Overfitting</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Training Stability (RP)</td>
<td>✓</td>
<td>Low variance</td>
<td>×</td>
<td>High variance</td>
</tr>
<tr>
<td>Sample Efficiency (RP)</td>
<td>✓</td>
<td>$&lt; 10^5$</td>
<td>×</td>
<td>$&gt; 10^6$</td>
</tr>
<tr>
<td>Intra-Trajectory Imitation (SL)</td>
<td>✓</td>
<td>Overfitting</td>
<td>×</td>
<td>GAN mode collapse</td>
</tr>
<tr>
<td>Inter-Trajectory Imitation (CH)</td>
<td>×</td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Training Disturbance Robustness (UE)</td>
<td>×</td>
<td>Distributional shift</td>
<td>✓</td>
<td>Mediocre</td>
</tr>
<tr>
<td>Testing Disturbance Robustness (NA)</td>
<td>×</td>
<td>Compounding error</td>
<td>✓</td>
<td>Mediocre</td>
</tr>
<tr>
<td>Clock Time</td>
<td>✓</td>
<td>$&lt; 1$ hour</td>
<td>×</td>
<td>$&gt; 10$ hours</td>
</tr>
</tbody>
</table>

Table 4.3.1: Summary of BC & GAIL Evaluations

Collectively, our results suggest that BC is in general favorable to GAIL. It outperforms GAIL in terms of both performance and imitation. While its robustness is less than desired, given the fact that BC is an offline method, its lack of robustness will not inflict real harm to the training agent. Furthermore, its batch nature and excellent efficiency means that it can be used as a great pre-training algorithm to warm-up other IL algorithms. For example, GAIL⁷. In contrast, GAIL’s unstable training is problematic in practice as GAIL needs to interacts with the environment.

Additionally, we keep track of the clock time for running BC and GAIL models on our machine⁸ and find that BC is much faster to train than GAIL, often in an order of magnitude. Combined with BC’s early convergence to expert performance, the amount of clock time BC requires to achieve good performance is in the minutes, making it more suitable for real-world domains where training time often presents a bottleneck for an embedded system [He et al., 2016].

Given our findings, it is natural to circle back to the existing literature and ask why is GAIL commonly thought to be better than BC? The results (Figure 3.1.1) from the original GAIL paper indeed do suggest that GAIL achieves higher performance than BC. How do our results reconcile with those in Ho and Ermon [2016]? It turns out the results in Ho and Ermon [2016] can be attributed to their non-standard

⁷We leave it to future work to explore the properties of such combination.
⁸See detailed hardware specification in Appendix C.
way of sub-sampling expert trajectories. For each expert trajectory $\tau$, Ho and Ermon [2016] do not use the full trajectory, but instead sub-sample and retain every 20 time steps starting with a random offset to construct a non-sequential trajectory, intended to make imitation more difficult [Kostrikov et al., 2019]. It is not surprising why BC performs worse in this setup. Due to its poor ability to generalize, BC cannot infer sensible actions to be taken for all the steps between the every-20 steps sampled. Then, its performance indeed severely degrades due to the compounding error issue, compared to that of a full-trajectory training setup.

Yet, it is unintuitive why such training methodology makes sense in the first place. There is no good reason to use sub-sampled version of expert trajectories when full trajectories are given, especially in safety-critical real-world domains (Section 2.3) where all available information of an expert policy should be used (i.e. full trajectories). In fact, the favorability of BC with full-trajectories is not entirely unnoticed in the literature. Kostrikov et al. [2019] state that BC achieves competitive results to GAIL in full-trajectory setting, but their notion of “competitiveness” is purely evaluated from a performance point of view, which we argued in Chapter 3 is not stand alone sufficient. Our results take a step further and argue that not only is BC competitive to GAIL, it is in fact favorable for all the reasons stated in this chapter.

4.4 Discussion

In this chapter, we put our proposed metrics in the last chapter to use by diagnosing BC and GAIL on a selected Mujoco environment. Our findings extend what is currently understood about these algorithms and therefore support the usefulness of the proposed evaluation metrics. They collectively challenge the common view in the literature that GAIL is preferable to BC. In particular, the results suggest that GAIL suffers greatly from training instability and inefficiency, which can be traced back to the root problems facing GAN and deep RL. In contrast, BC is stable and efficient, but is not robust to disturbance at either training or testing time.

Yet, to our best knowledge, BC remains the only general purpose batch imitation learning (BIL) algorithm. But that leaves open the possibility of new BIL algorithms that reach beyond a reduction of IL to supervised learning, and are robust to trajectory degradation, which BC is not. It is worth noting that the existence of a robust BIL algorithm is not only interesting from an academic perspective, but from a practitioner point of view. Such an algorithm means that a good policy can be learned in an offline manner while makes little assumption about the optimality of the trajectories. And these two properties are desirable in real world, where data collection is messy and environmental interaction expensive. For these reasons, we now devote the rest of this thesis working towards the goal of devising a new BIL
algorithm that is competitive to BC while improving robustness.

In the next chapter, we begin our exploration by surveying the batch reinforcement learning (BRL) framework as well as a state-of-art BRL technique Batch Constrained Q-Learning (BCQ) Fujimoto et al. [2018a]. Inspired by GAIL’s innovative synthesis of generative modeling and deep RL, we aim to similarly design a BIL algorithm by leveraging recent advances in (deep) BRL.
GAIL builds upon recent advances in deep reinforcement learning to perform imitation learning in high-dimensional tasks. However, its practicality is limited by its instability and sample inefficiency inherited from on-policy policy gradient RL algorithms. A safer RL approach is batch reinforcement learning (BRL), in which a policy is learned in a complete off-line fashion using a large batch of transition samples in the environment.

Can we convert a BRL algorithm into a BIL algorithm? An affirmative answer would indeed be appealing as BIL algorithms can circumvent the need for accessing the environment as well as the unpredictable nature of GAIL training. To begin answering this question, we devote this chapter to an introduction to BRL and a state-of-art BRL algorithm, batch-constrained Q-learning (BCQ) [Fujimoto et al., 2018a].

5.1 Batch Reinforcement Learning Preliminaries

As in the standard RL setup, BRL [Lange et al., 2012] aims to learn a behavior policy that maximizes the cumulative rewards in the environment. However, the learning procedure does not have access to the environment by taking actions and observing feedbacks. Instead, a policy is directly learned using a
dataset (batch) of transition samples from the environment:

\[ \mathcal{B} := \{(s_i, a_t, s'_i, r_i)_{i=1}^N\} \]

In the most general case, there is no assumption made about the samples in \( \mathcal{B} \). They can be from a set of trajectories or sampled from the environment by random exploration. In particular, the dataset is not assumed to have come from an expert policy \( \pi^* \). See Lange et al. [2012] for a detailed treatment of the learnability of an optimal policy under different assumptions for \( \mathcal{B} \).

Compared to standard reinforcement learning, batch reinforcement learning does not enjoy the luxury of interacting in the environment. However, this seemingly harsh restriction is in fact more realistic in many sequential decision making settings in which reinforcement learning algorithms are applicable in learning good policies [Jiang and Li, 2015] [Achiam et al., 2017]. These settings share the common feature that interacting with the environment is either impossible \(^1\) or risky, making it essential that learning is done off-line.

One common approach to batch reinforcement learning is to perform the update rule of Q-learning\(^2\) in a batch, supervised setting, otherwise known as fitted-Q iterations (FQI) [Ernst et al., 2005]. We provide a brief exposition of FQI as it is a foundational approach in BRL and shared the core algorithmic idea of performing supervised Q-learning as algorithms presented later in this chapter.

### 5.1.1 Fitted Q-Iteration

In FQI, the optimal Q-function is estimated iteratively using regression and the batch \( \mathcal{B} \). At each iteration, the algorithm fits the batch state inputs to their corresponding Q-value using the regression algorithm of choice. The regression’s output is then used to construct the target Q-value for the next iteration. This iterative procedure continues until a stopping condition (e.g. convergence) is reached. The pseudocode of the algorithm is provided below.

\(^1\) though this limitation can be circumvented by building a simulator of the environment [Tan et al., 2018], but not all environments can be simulated

\(^2\) We refer readers to [Sutton and Barto, 2018] for an excellent introduction to Q-learning.
Algorithm 2 Fitted Q-Iteration

Require: Batch \( B = \{ (s_i, a_i, s_{i+1}, r_i) \}_{i=1}^{N} \), a regression algorithm \( R \)

1: Initialize \( T = 0 \) and \( \hat{Q}_N \) to be a function equal to \( 0 \) everywhere.
2: while Stopping condition not reached do
3: \( N = N + 1 \)
4: Build the training set \( S = \{ (x_i, y_i) \}_{i=1}^{N} \) where
\[
x_i = (s_i, a_i), y_i = r_i + \gamma \max_{a \in A} \hat{Q}_{N-1}(s_{i+1}, a)
\]
5: Use the regression algorithm \( R \) to estimate \( \hat{Q}_N \) from \( S \)
6: end while

Note that a max operation is taken in FQI, and this is only computationally tractable in discrete, tabular environments where the Q-value of each action given the state can be enumerated or if we enforce convexity assumption on the Q-function itself. In general, this limitation is circumvented using kernel-based methods or other function approximation techniques \cite{Ernst2005, Riedmiller2005}; 5.2 Batch-Constrained Deep Q-Learning

Now, we study batch-constrained deep Q-learning (BCQ), a state-of-art batch RL algorithm, that serves as the bedrock of our method. At its core, BCQ learns a policy much like FQI, by iteratively fitting approximations of the optimal Q-function with respect to the batch data \( B \). Unlike previous BRL methods, BCQ is effective in tackling the problem of distributional mistatch, etc, etc by incorporating techniques recently introduced to stabilize reinforcement learning and deep learning algorithms.

BCQ aims to learn a policy with a similar state-action visitation distribution as the empirical distribution in \( B \). To measure the similarity of a given pair \((s, a)\) to transitions in \( B \), BCQ defines a learned state-conditioned marginal likelihood \( P_B(a|s) \). This likelihood measure can be interpreted as the probability that action \( a \) was taken at state \( s \) under the policy \(^3\) that generates \( B \). Hence, a desirable policy needs to maximize \( P_B(a|s) \), in order to reduce instances of entering unfamiliar state-action pairs that may incur compounding error, a phenomenon that we first described in Chapter 2. Since \( \text{arg max}_a P_B(a|s) \) is intractable in high-dimensional continuous domains, BCQ trains a parametric generative model of the batch, \( G_\omega(s) \), represented by a variational auto-encoder (VAE)\(^4\) \cite{Kingma2013}, \( G_\omega = \{ E_\omega, D_\omega \} \). This VAE models a Gaussian latent space, using which action samples at a state can be

\(^3\)Or more generally, mixture of policies since we do not make assumption about the policy sources of \( B \).

\(^4\)See details in Appendix E.
drawn to approximate $\arg\max_a P_G^T(a|s)$.

Then, for a given state, BCQ generates plausible actions according to the conditional likelihood measure, and selects the action with the highest Q-value, determined by a Q-network $Q_\theta$ jointly trained with the VAE. In order to increase the policy’s robustness to sub-optimal actions unseen in the batch, DRBCQ increases the diversity of actions seen during training by incorporating a perturbation model $\xi_\varphi(s, a, \Phi)$, represented by some parametric distribution restricted to the range of $[-\Phi, \Phi]$. Then, the algorithm trains for a policy that is optimal after the actions are perturbed. In summary, the resulting policy is

$$
\pi(s) = \arg\max_{a_i + \xi_\varphi(s, a_i, \Phi)} Q_\theta(s, a_i + \xi_\varphi(s, a_i, \Phi)), \{a_i \sim G_\omega(s)\}_{i=1}^n \tag{5.1}
$$

The perturbation model itself is also trained to maximize $Q_\theta$ using the deep deterministic policy gradient algorithm (DDPG) [Silver et al., 2014] by sampling $a \sim G_\omega(s)$:

$$
\varphi \leftarrow \arg\max_{\varphi} \sum_{(s,a) \in \mathcal{B}} Q_\theta(s, a + \xi_\varphi(s, a, \Phi)) \tag{5.2}
$$

As for the training of $Q_\theta$, stabilization techniques introduced to improve the performance of deep Q-network (DQN) [Mnih et al., 2015] [Van Hasselt et al., 2016] [Fujimoto et al., 2018b] are used. We present the entire algorithm below (reproduced from [Fujimoto et al., 2018a]).

**Algorithm 3** Batch Constrained Deep Q-Learning

**Require:** Batch $\mathcal{B}$, horizon $T$, target network update rate $\tau$, mini-batch size $N$, max perturbation $\Phi$, number of sampled actions $n$, minimum weighting $\gamma$.

1: Initialize Q-networks $Q_{\hat{\theta}}, Q_{\theta}$, perturbation network $\xi_\varphi$, and VAE $G_\omega = \{E_\omega, D_\omega\}$, with random parameters $\theta_1, \theta_2, \varphi, \omega$, and target networks $Q_{\hat{\theta}}', Q_{\theta}'$, $\xi_{\varphi}'$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \varphi' \leftarrow \varphi$

2: for $t=1,\ldots,T$ do

3: Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$

4: $\mu, \sigma = E_\omega(s, a), \bar{a} = D_\omega(s, z), z \sim \mathcal{N}(\mu, \sigma^2)$

5: $\omega \leftarrow \arg\min_\omega \sum (a - \bar{a})^2 + D_{KL}(\mathcal{N}(\mu, \sigma^2)\|\mathcal{N}(0, 1)))$

6: Sample $n$ actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$

7: $r + \gamma \max_a [\lambda \min_{j=1,2} Q_{\theta'_j}(s', a_i) + (1-\lambda) \max_{j=1,2} Q_{\theta'_j}]$

8: $\theta \leftarrow \arg\min_\theta \sum (r - Q_\theta(s, a))^2$

9: $\varphi \leftarrow \arg\max_{\varphi} \sum Q_{\hat{\theta}}(s, a + \xi_\varphi(s, a, \Phi)), a \sim G_\omega(s)$

10: Update target networks: $\theta'_1 \leftarrow \tau \theta + (1-\tau)\theta'_1, \varphi' \leftarrow \tau \varphi + (1-\tau)\varphi'_1$

11: end for

In practice, BCQ is proven to be highly efficient with respect to training iterations and robust. It is able to learn a good policy with different levels of batch data quality in numerous Mujoco environments while using the same set of parameters throughout. We refer interested readers to Fujimoto et al. [2018a] for
more thorough details about this algorithm.

5.3 Discussion

In this chapter, we review BRL and Batch Constrained Q-Learning (BCQ), a state-of-art deep BRL algorithm. The excellent performance, efficiency, and robustness of BCQ makes it an appealing candidate to be the RL sub-routine of a BIL algorithm, much like how GAIL builds on top of PPO. The main obstacle is the need of a reward function for BCQ without environmental interactions. In the next chapter, we explore a self-supervised reward learning approach that is compatible with BCQ, and then present a new BIL algorithm, Disagreement-Regularized Batch-Constrained-Q Imitation Learning (DRBIL).
If intelligence was a cake, self-supervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake.

Yann LeCun

6

Disagreement Regularized Batch-Constrained-Q Imitation Learning

In this chapter, we present our novel batch imitation learning algorithm, Disagreement-Regularized Batch-Constrained-Q Imitation Learning (DRBIL), which combines BCQ with a self-supervised reward function introduced in Brantley et al. [2020] that makes it suitable for IL. We compare DRBIL to BC, and show that it preserves the same level of stability and efficiency, while attaining robustness to unreliable expert trajectories, which BC is not able to do. For completeness, we also evaluate DRBIL using the metrics introduced in chapter 3. Additionally, we obtain a surprising result that BCQ paired with random rewards, which we call Random Batch-Constrained-Q Imitation Learning (RBIL), also achieves expert performance. Consistent with the main theme of this thesis, RBIL's effectiveness speaks to the greater scrutiny required to validate existing algorithms (including ours).

6.1 Self-Supervised Reward Function for Batch Imitation Learning

Our goal is to design a new BIL that builds on top of a batch RL algorithm, and in particular, BCQ. However, the batch data in BRL settings come with reward that is associated with each transition, which does not exist in an IL setting. Therefore, to unlock the potential of BCQ to IL, we need a method to assign reward to each of our expert transitions.
This is difficult because BIL does not access more samples from the environment, so previous reward function used for IL, such as GAIL’s discriminator reward, do not apply. Hence, a BIL algorithm that leverages RL in any form requires a reward function that is directly computable from the expert batch, in a self-supervised manner.

What kinds of feature should such a batch reward function satisfy? To answer this question properly, we crystallize the desirable properties of a BIL algorithm. First, due to its lack of ability to take actions in the environment, a BRL policy cannot afford to outputting risky actions due to the lack of environmental feedback. As such, a BRL policy should output action(s) that are similar to the ones seen in the expert batch conditioning on the state. Satisfying this criteria implicitly ensures that the policy avoid actions that are likely to lead to states unseen or underrepresented in the expert batch, hence reducing the risky of compounding error.

Yet, not all state-action pairs in the batch are equal. If the expert trajectories are allowed to be sub-optimal sometimes, then naturally some transitions in the batch are less desired than others. Even if all trajectories are optimal, there are transitions that are more important than others. Therefore, an informative reward signal in the batch setting needs to differentiate the essential and safe transitions in the batch from the ones that either do not satisfy these criteria or only do so weakly. Note that despite BC’s excellence performance when the expert trajectories are optimal, its performance degradation in mixed setting can be attributed to its inability to satisfy the desiderata described above.

So, now the question is: how do we achieve this goal?

6.1.1 Uncertainty Cost

Brantley et al. [2020] introduce a self-supervised imitation learning (cost) signal, named Uncertainty Cost (UC). The intuition behind this method is that essential (e.g. frequently included) transitions will be included more frequently in bootstrapped replications of the expert batch $D$. Thus, if we train a supervised classifier (e.g. Behavior Cloning) on each of the bootstrapped samples of $D$, the variance of the likelihood of a state-action pair is lower for pairs that are essential in $D$ and higher otherwise. This variance measure can then be directly used as a signal for RL. More formally,

$$
C_U(s, a) \triangleq \text{Var}_{\pi \sim p(\pi|D)}(\pi(a|s))
$$

(6.1)

where $\pi \sim p(\pi|D)$ is the posterior distribution of (Behavior Cloning) policies learned from the expert trajectories. This posterior distribution is estimated using an ensemble of policies $E = \{\pi_e\}_{e=1}^{|E|}$ with
different initializations and trained on different bootstrapped samples of $\mathcal{D}$ as described above.

This loss incurs high value (i.e. high variance) when (1) the policies from the posterior do not agree on the action because the action is sparsely presented in the expert trajectories even if the state is well represented or (2) the state itself is unfamiliar to the policies, and thus consequently any action will incur high variance over the posterior. As such, minimizing this loss encourages the agent to visit regions of high coverage in the expert trajectories.

To validate the Uncertainty Cost’s efficacy\(^1\), we construct an expert dataset $\mathcal{D}$ on the Walker2d-v2 environment by including five expert trajectories (labeled as 0,1,2,3,4) and five sub-optimal trajectories (labeled as 95, 96, 97, 98, 99), using the trajectory generation procedure described in 4.1. Then, we train an ensemble of five BC models using five independent bootstrapped samples of $\mathcal{D}$. Using the ensemble, we compute the Uncertainty Cost of each transition in the bootstrapped samples and plot the boxplot of Uncertainty Costs for each individual trajectory.

![Walker2d-v2 Uncertain Cost Boxplot](image)

**Figure 6.1.1:** Poorer trajectories incur higher Uncertainty Cost.

As shown, expert trajectories incur significantly lower Uncertainty Costs than degraded trajectories. Furthermore, they exhibit lower variance, judged by the length of the boxes. This validates the hypothesis in Brantley et al. [2020] that Uncertainty Cost is able to discriminate good versus degraded transitions in the expert dataset.

\(^1\)this verification experiment is omitted in the original paper.
6.2 The Full Algorithm

In this section, we present DRBIL in full. DRBIL is conceptually simple: it extends BCQ to the imitation learning setting by incorporating Uncertainty Cost as its reward signal. Because the Uncertainty Cost is computed in an off-line mode, DRBIL is a complete batch-mode imitation learning that does not require further sample collection from the environment. For completeness, we present the pseudocodes of the UC subroutine and DRBIL below.

**Algorithm 4 Uncertainty Cost**

**Require:** Expert demonstration data \( \mathcal{D} = \{ (s_i, a_i) \}_{i=1}^{N} \)
1. Initialize policy \( \pi \) and policy ensemble \( E = \{ \pi_e \}_{e=1}^{E} \)
2. for \( e = 1, \ldots, E \) do
3. Sample \( \mathcal{D}_e \sim \mathcal{D} \) with replacement, with \( |\mathcal{D}_e| = D \)
4. Train \( \pi_e \) to minimize \( J_{BC}(\pi_e) \) on \( \mathcal{D}_e \) to convergence
5. end for

**Algorithm 5 Disagreement-Regularized Batch-Constrained-Q Imitation Learning**

**Require:** Batch \( \mathcal{B} \), horizon \( T \), target network update rate \( \tau \), mini-batch size \( N \), max perturbation \( \Phi \), number of sampled actions \( n \), minimum weighting \( \gamma \).
1. Initialize Q-networks \( Q_{\theta_1}, Q_{\theta_2} \), perturbation network \( \xi_{\varphi} \), and VAE \( G_{\omega} = \{ E_{\omega}, D_{\omega} \} \), with random parameters \( \theta_1, \theta_2, \varphi, \omega \), and target networks \( Q_{\theta_1}', Q_{\theta_2}', \xi_{\varphi}' \) with \( \theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \varphi' \leftarrow \varphi \)
2. for \( t = 1, \ldots, T \) do
3. Sample mini-batch of \( N \) transitions \( (s, a, r, s') \) from \( \mathcal{B} \)
4. \( \mu, \sigma = E_{\omega}(s, a), \bar{a} = D_{\omega}(s, z), z \sim \mathcal{N}(\mu, \sigma^2) \)
5. \( \omega \leftarrow \arg\min_{\omega} \sum (a - \bar{a})^2 + D_{KL}(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0, 1)) \)
6. Sample \( n \) actions: \( \{ a_i \sim G_{\omega}(s') \}_{i=1}^{n} \)
7. \( -C_U(s, a) + \gamma \max_{a_i} [\lambda \min_{j=1,2} Q_{\theta_j}'(s', a_i) + (1 - \lambda) \max_{j=1,2} Q_{\theta_j}'] \)
8. \( \theta \leftarrow \arg\min_{\theta} \sum (y - Q_{\theta}(s, a))^2 \)
9. \( \varphi \leftarrow \arg\max_{\varphi} \sum Q_{\theta}(s, a + \xi_{\varphi}(s, a, \Phi)), a \sim G_{\omega}(s) \)
10. Update target networks: \( \theta_j' \leftarrow \tau \theta + (1 - \tau) \theta_j', \varphi' \leftarrow \tau \varphi + (1 - \tau) \varphi_j' \)
11. end for

6.3 Results

We present experimental results of DRBIL on three Mujoco environments: Walker2d, Hopper, and Humanoid². In this section, we primarily present results on the Walker2d, as before. The whole results for Hopper and Humanoid are included in appendix D. Our experiments aim to answer the following questions:

²An graphical illustration of these three environments are included in Appendix A.
• How does DRBIL compare to BCQ?
• Is DRBIL a good batch imitation learning algorithm?
• How does DRBIL compare to BC?

Comparing DRBIL directly with BCQ will inform us how much worse it is to use a self-supervised reward function (DRBIL) compared to the real reward function that comes from the environment (BCQ), while keeping the model architecture same. Because BCQ receives the actual rewards from the environment, we expect that it performs better than DRBIL. But if DRBIL performs comparably, then it indicates that Uncertainty Cost is good enough for learning from a batch dataset, which serves our purpose of imitation learning. The second question and third question answers if DRBIL is a desirable BIL algorithm and can improve upon BC, the competitive baseline BIL algorithm. To facilitate this comparison, DRBIL is evaluated using the metrics proposed in Chapter 3, and its results are compared with that of BC.

6.3.1 DRBIL vs. BCQ

We compare DRBIL with BCQ in the Mujoco environments and report their running performance as well as best training performance achieved. The same set of model parameters are used for both algorithms in all environments. See appendix C for the full model specifications. All models are also trained using the $D^\text{expert}$ dataset (hence label “BCQ - good” in the plot for BCQ) for $10^6$ iterations with a batch size of 100 (the original hyperparameters in Fujimoto et al. [2018a]). Each model is trained using five random seeds, and the average and one standard deviation of performance are displayed in Figure 6.3.1.
Our results suggest that DRBIL achieves competitive performance to BCQ. The running performances reveals that DRBIL, like BCQ, achieves fast convergence, and in some cases (e.g. Hopper), exhibits lower variance during training. Though its performance is inferior to that of BCQ in Hopper and Walker2d, DRBIL achieved superior performance in Humanoid, until the very end where DRBIL slips in performance due to potential overfitting. These findings altogether speak to the viability of using a self-supervised reward signal to convert a BRL algorithm into a competitive BIL algorithm.

Now, it remains to show that DRBIL compares well against BC, with respect to the new set of metrics proposed in chapter 3. Note that in chapter 4, we conclude that BC performs adequately on all metrics except for the robustness ones, so it is natural to investigate if DRBIL can improve upon BC in these scenarios.

6.3.2 DRBIL vs. BC

As before, the same set of parameters are used for both algorithms in all environments. Because DRBIL is computationally more demanding to train than BC, it is infeasible to train with the full expert dataset in each iteration. Instead, it is trained for 20000 iterations with a batch size of 100, which our experiments...
show is sufficient for convergence. BC is trained for 10000\(^3\) iterations with also a batch size of 100\(^4\). All other experimental protocols are kept same. The full experimental results\(^5\) are reported below.

6.3.2.1 Best Performance

![Graphs showing BC and DRBIL performance](image)

Table 6.3.1: DRBIL & BC Mujoco Best Performances (Batch Size 100).

<table>
<thead>
<tr>
<th>Model</th>
<th>1 Trajectory</th>
<th>3 Trajectories</th>
<th>5 Trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC(_{good})</td>
<td>3116.77±245.89</td>
<td>3682.76±242.02</td>
<td>3751.40±58.10</td>
</tr>
<tr>
<td>DRBIL(_{good})</td>
<td>3633.72±294.41</td>
<td>3680.59±230.65</td>
<td>3921.91±189.40</td>
</tr>
<tr>
<td>BC(_{mixed})</td>
<td>2582±596.82</td>
<td>3086.03±471.25</td>
<td>3230.92±286.07</td>
</tr>
<tr>
<td>DRBIL(_{mixed})</td>
<td>2586.10±193.32</td>
<td>3149.12±421.74</td>
<td>3223.05±262.89</td>
</tr>
</tbody>
</table>

As shown, for both pure and mixed expert trajectories, DRBIL and BC achieve comparable training performance. This suggests that despite its complicated structure, DRBIL is not difficult to train. Specifically, in the best case scenario, DRBIL does not underperform BC.

It is worth noting that the results here for BC\(_{mixed}\) is significantly better than those in Chapter 4 (Figure 4.2.1), despite the models being trained with only 100 samples in each update as opposed to the full dataset. This suggests that when the batch size is small, there is only a small chance that sub-optimal transitions are sampled in each iteration of update. Hence, the policy after each update more or less mimics the policy as if it were trained with pure expert trajectories, explaining why BC\(_{mixed}\) and BC\(_{good}\) are comparable in performance under this metric. In contrast, when BC is trained with the full dataset in

---

\(^3\)Because BC’s performance converges quickly, our experiments with 20000 iterations did not make a visible difference in performance.

\(^4\)BC was trained with full batch in chapter 4.

\(^5\)In all plots, DRBIL is referred to as DRBCQ.
each iteration as in Chapter 4, it cannot avoid learning from the degraded transitions and hence obtain poor performance.

6.3.2.2 Running Performance

![Figure 6.3.2: BC & DRBIL Walker2d Running Performances](image)

Each algorithm’s training curves for $D^s_{\text{expert}}$ and $D^s_{\text{mixed}}$ are plotted against each other. These two graphs provide convincing evidence that DRBIL is more robustness to BC in terms of the quality of expert trajectories.

While the two models are comparable in terms of the best performance they achieve during training, comparing their evolutions during training immediately shows the stability of DRBIL compared to the instability of BC in the small batch size regime. Training with a small batch size exposes BC’s inclination to overfitting to its batch training set. This is not a serious problem when all the training samples are optimal, but if some of them are degraded (in our experiment, by design), then it is highly likely that BC overfits to some poor samples in one iteration and hence dives into a local optimum, from which the model never recovers. This is indeed what the graph shows empirically. While the model is steadily improving initially, its performance quickly declines and never recovers. This phenomenon is robust across five seeds, demonstrating that it is a serious problem facing BC to be reckoned with.

In sharp contrast, the presence of degraded trajectories does not significantly hamper DRBCQ’s performance at all. The model still steadily improves, though the performance is asymptotically lower than the pure expert trajectories setting. This is not surprising because the policy

$$
\pi(s) = \arg \max_{a_{i} + \xi \phi(s, a_{i}, \Phi)} \{ Q_{\theta}(s, a_{i} + \xi \phi(s, a_{i}, \Phi)) \}^{n}, \{ a_{i} \sim G_{w}(s) \}_{i=1}^{n}
$$

is designed to optimize for the performance when actions are perturbed. As such, DRBIL learns robust
policy that stably improves in presence of mixed trajectories.

6.3.2.3 Trajectory Stepwise Likelihood

The likelihood metric is not included because it does not apply to DRBIL, which does not parameterize the learned policy as a state-conditional action distribution.

6.3.2.4 Cumulative-Statistics Histogram

Figure 6.3.3: Histograms of Cumulative Rewards and Durations of Expert and Imitator Trajectories in Walker2d-v2.
The distributions over trajectory cumulative rewards and durations are not informative. There does not exist qualitatively different patterns between BC and DRBIL. This is to be expected since DRBIL is a model built to improve the robustness of batch-mode learning, and the original BCQ model is a RL model which does not account for the notion of imitation at all. Thus, there is no reason to expect that our simple modification to BCQ to make it compatible for IL can improve the quality of imitation.

6.3.2.5 Noisy Actions

For Noisy Actions, trained models are executed with action disturbance as in Chapter 4, and ten trajectories are collected for each model. The graphical and numerical results are reported below.

![Graphs showing Noisy Actions Evaluations](image)

**Figure 6.3.4:** Walker2d Noisy Actions (NA) Evaluations

<table>
<thead>
<tr>
<th>Model</th>
<th>1 Trajectory</th>
<th>3 Trajectories</th>
<th>5 Trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC, $\varepsilon = 0$</td>
<td>1894.77±232.57</td>
<td>1853±252.12</td>
<td>1912±255.30</td>
</tr>
<tr>
<td>DRBIL, $\varepsilon = 0$</td>
<td>1971.21±173.89</td>
<td>2038.43±193.63</td>
<td>1994.58±189.11</td>
</tr>
<tr>
<td>BC, $\varepsilon = 0.3$</td>
<td>≈ 350 (random)</td>
<td>≈ 350</td>
<td>≈ 350</td>
</tr>
<tr>
<td>DRBIL, $\varepsilon = 0.3$</td>
<td>≈ 350 (random)</td>
<td>≈ 350</td>
<td>≈ 350</td>
</tr>
</tbody>
</table>

**Table 6.3.2:** Walker2d Noisy Actions Evaluation Table

When the only source of disturbance comes from a white noise added to the models’ action outputs
(ε = 0), both BC and DRBIL observe considerable decline in performance. Their relative performances are also comparable, though DRBIL seems to slightly outperform. But we refrain from drawing any conclusion because the differences are too small, and may be entirely due to random chance.

However, we did expect DRBIL to demonstrably outperform BC in this testing regime because DRBIL is trained to be robust to action disturbance at the training time. Understanding why it demonstrates excellent robustness during the training time, but not testing time is worth further investigation in future work.

On the other hand, when there is also a positive probability ε that a random action is taken at each step (ε = 0.3), both models fail to perform above random, suggesting that this source of disturbance is too difficult for state-of-art batch IL algorithms to learn at all.

In conclusion, our findings in this test are suggest that despite DRBIL’s robustness to trajectory degradation during the training time, its performance when evaluated using Noisy Actions mechanisms does not outperform that of BC, contrary to our prior expectation.

6.3.2.6 Summary of Results

As in chapter 4, we summarize the comparison between DRBIL and BC in a table. Both algorithms operate in batch mode and achieve competitive performance. The main difference is that, by design, DRBIL is more robust to trajectory degradation and environmental disturbance. However, DRBIL takes longer to train and requires twice as many updates to achieve the same asymptotic level of performance. In conclusion, DRBIL is comparable to BC, and is more robust to mixed trajectories⁶, making it a viable batch IL algorithm given that computation is not a source of constraint. However, it does not outperform BC along many other axes that are considered, suggesting much room for improvement.

⁶Additional experiments in appendix D weaken this claim.
<table>
<thead>
<tr>
<th>Property</th>
<th>DRBIL</th>
<th>BC</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Performance (BP)</td>
<td>✓</td>
<td>✓</td>
<td>DRBIL slightly better</td>
</tr>
<tr>
<td>Training Stability (RP)</td>
<td>✓</td>
<td>×</td>
<td>BC suffers from overfitting</td>
</tr>
<tr>
<td>Sample Efficiency (RP)</td>
<td>✓</td>
<td>✓</td>
<td>BC is 2x more sample efficient</td>
</tr>
<tr>
<td>Intra-Trajectory Imitation (SL)</td>
<td>NA</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Inter-Trajectory Imitation (CH)</td>
<td>?</td>
<td>?</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>Training Disturbance Robustness (UE)</td>
<td>✓</td>
<td>×</td>
<td>DRBIL robust to mixed dataset</td>
</tr>
<tr>
<td>Testing Disturbance Robustness (NA)</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Clock Time</td>
<td>×</td>
<td>✓</td>
<td>DRBIL ≈ 3 hours, BC &lt; 1 hour</td>
</tr>
</tbody>
</table>

Table 6.3.3: Summary of DRBIL & BC Evaluations

6.4 Reward Agnostic Imitation Learning

Now, let’s loop back to the comparison between DRBIL and BCQ. Though both algorithms are shown to have comparable performances, does it necessarily mean that it is the Uncertainty Cost that allows the transformation from BCQ to DRBIL, so that it can repurposed for imitation learning?

The mere observation that DRBIL and BCQ achieve similar performances can be interpreted in two ways. First, the Uncertainty Cost used to provide the reward in DRBIL achieves its intended effect, and is comparable to the true environmental reward in steering steady policy improvement. This interpretation is indeed what we claimed in the DRBIL vs. BCQ comparison above.

A second interpretation is that the architecture of BCQ itself is already enough to learn a good policy, when trained with expert trajectories, and that the particular reward signal does not matter. If this is the case, then indeed it can also explain the results that were obtained.

In this section, we investigate this interpretation, and find the surprising result that the reward signal fed into BCQ, suitably modified to make use of expert trajectories, can be entirely randomized - the performance is competitive with that of both BCQ and DRBIL. And we name this model as Randomized Batch-Constrained-Q Imitation Learning (RBIL). Note that RBIL, like DRBIL, is an imitation learning algorithm because it learns using a batch of expert trajectories, and its reward function trivially does not depend on the environment.

RBIL can be trivially implemented on top of BCQ. Instead of using the rewards from the environment, RBIL replace them simply by samples drawn from an uniform distribution between [0, 1]. In the table
below, we illustrate the differences among BCQ, DRBIL, RBIL.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Purpose</th>
<th>Reward Function</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCQ</td>
<td>RL</td>
<td>Environment</td>
<td>BCQ</td>
</tr>
<tr>
<td>DRBIL</td>
<td>IL</td>
<td>Uncertainty Cost</td>
<td>BCQ</td>
</tr>
<tr>
<td>RBIL</td>
<td>IL</td>
<td>$U \sim [0, 1]$</td>
<td>BCQ</td>
</tr>
</tbody>
</table>

Table 6.4.1: RBIL, DRBIL, BCQ Differences

We compare RBIL's running performance on three Mujoco environments with those of BCQ and DRBIL, when the models are trained with either pure expert trajectories (left) or mixed trajectories (right). We are interested not only in RBIL's comparison with DRBIL but also with BCQ because the pairwise comparisons investigate two different hypotheses. Comparing RBIL and DRBIL investigates whether a well curated (self-supervised) reward function is needed to achieve robust BIL, while comparing RBIL and BCQ investigates whether the algorithm architecture of BCQ alone is enough to learn a good policy. Both questions are interesting in their own rights, which we study by plotting the training curves for all three algorithms together for each environment and each type of dataset below.
Figure 6.4.1: RBIL ("BCQ - random") (green) achieves competitive performance on the Mujoco environments.

As shown, all three variants of BCQ achieve similar results on the three environments. RBILL (green), in particular, appears to outperform DRBIL and is on par with BCQ. While this surprising result weakens our claim that DRBIL’s performance comes from the usefulness of the Uncertainty Cost as an imitation signal, it highlights the importance of rigorous evaluation of IL algorithms (including our own), which is a central theme of this thesis, as well as the viability of converting BRL algorithms into BIL algorithms. Perhaps, a self-supervised reward function is not needed, and we can fully rely on the BRL components themselves to achieve successful imitation. And we leave to future work to investigate why RBIL is able to perform so well, which will bring us closer to truly understanding the effectiveness of BCQ in the first place.
Discussions & Future Directions

In this chapter, we discuss key takeaways of this thesis and synthesize them with future directions of research.

7.1 Takeaways and Future Work

The insights of this thesis fall under two main categories: (1) the need for more rigorous science for imitation learning, and (2) the promise of batch imitation learning.

7.1.1 The Need for Rigorous Science for Imitation Learning

One of the main contributions of this thesis is its benchmarking for baseline IL algorithms (BC & GAIL), which arrive at a conclusion that challenges the common viewpoint in the literature. The novelty of our evaluation effort itself speaks to the need for a stronger empirical foundation for imitation learning research. The favorable results for GAIL vs BC in the literature Ho and Ermon [2016] seem to result from an unusual training setup. In contrast, in this work, all experimental setups are described and explained in detail.

The lack of rigorous evaluation metrics cast doubt on the validity of stated improvement that newer algorithms attain over existing algorithms. For one, this work shows that the common view that GAIL is
favorable to BC may need to be reevaluated. Further, though Uncertainty Cost is shown to discriminate good and bad trajectories in the batch data, the experiment that proves this, which we include in Chapter 6, was missing from the original work Brantley et al. [2020]. Moreover, since much of the recent progress on IL builds upon GAIL [Kostrikov et al., 2019] [Blondé and Kalousis, 2018] [Sasaki et al., 2018], a rigorous vetting process is required to validate these methods due to recent challenges to the validity of GAN improvements as well as the popular PPO algorithm [Engstrom et al., 2020] [Ilyas et al., 2020], which the thesis uses to train the expert policies.

As such, a stricter standardization of reporting and evaluation in IL is crucial for advancing the field. Our thesis makes an attempt with a set of six simple metrics. They are by no means exhaustive, but are able to discover new insights that are either ignored or understudied in the current literature. We hope that this work may inspire extensive future work in benchmarking imitation learning algorithms. In particular, we believe that more sophisticated methods for evaluating imitation learning can be important. Some of the methods in this work, Stepwise Likelihood and Cumulative-Statistics Histogram, though providing valuable insights, do not elucidate in themselves ways to improve algorithms. More quantitative metrics such as KL-divergence may be useful and provide a direct path to better optimization for imitation. Similarly, more robustness metrics can also be proposed. For example, a concern with the evaluation in Ho and Ermon [2016] is with its experimental design of 20-step sub-sampled expert trajectories. However, a robustness test measuring algorithm performance under n-step sub-sampled expert trajectories (for small n) may be valuable, as it measures the algorithm’s robustness to missing data in expert trajectories.

Finally, progress can also be made by building directly on top of our method by replicating the body of Chapter 4 on more testing environments. Though the present findings are intuitive and are relatively robust, we do not exclude the possibility that the particularity of the environment we chose is the main cause of the observed findings. Only with more systematic and extensive benchmarking, we can be reliably sure of our conclusions.

7.1.2 The Promise of Batch Imitation Learning

A direct result of our benchmarking effort is that BIL is a viable and promising learning framework compared to the pure RL based framework, which includes GAIL. Combining recent progress in BRL and self-supervised learning, we are able to devise a novel algorithm DRBIL that demonstrates competitive performance and outperforms other methods when the expert dataset is of mixed quality. As such, despite the necessity of a self-supervised reward function being challenged by our discovery of RBIL, DRBIL still presents an innovation in itself. It showcases the viability of combining previously two independent fields of deep learning together to solve a related but different problem of IL, and opens door
for much future research.

A natural next step is to explore the viability of other state-of-art BRL algorithms Kumar et al. [2019] Agarwal et al. [2019] as well as self-supervised learning techniques Pathak et al. [2017] Pathak et al. [2019]. Some of these self-supervised methods may not be directly applicable to the imitation learning setting, but exploring such possibility is indeed an interesting research direction. In doing so, the following questions are of particular interests:

1. What properties of BRL algorithms are suitable for BIL?
2. What properties of self-supervised reward function are suitable for BIL?
3. How to best combine a BRL algorithm with a reward function to perform BIL?

Additionally, recall that our ablation study at the end of chapter 6 demonstrates a surprising result that RBIL is enough to achieve expert performance, and often does even better. This suggests that the magic may be entirely due to the regularization of BCQ itself. Gaining a deeper understanding of this phenomenon is of both theoretical and practical interests.

Again, the discovery of RBIL highlights the crucial missing step of algorithm validation in the original BCQ paper. It prematurely draws the conclusion that BCQ’s algorithmic innovation is the source of its training success, without rigorous ablation study that isolates the effect of individual parts. Others have also expressed concern about evaluation in the deep learning literature Marcus [2018] And our thesis, adding to the sentiment that the field is moving too quickly, demonstrates the danger of taking existing algorithms and drawing conclusions at their face values without critical introspection.
In this thesis, we consider the problem of imitation learning. Picking up where the literature leaves off, we advance imitation learning evaluation and batch imitation learning. Our proposed evaluation metrics for imitation learning are effective and offer new insights into IL algorithms. In particular, they provide convincing results that challenge the common view in the literature that GAIL is superior to BC. Additionally, our robustness study uncovers both algorithms’ sensitivity to degraded data. To remedy this issue, we combine methods from batch reinforcement learning and self-supervised learning to arrive at a batch imitation learning algorithm, Disagreement-Regularized Batch-Constrained Q-Learning, which is robust to data degradation.

This thesis, as a whole, calls for greater attention from the imitation learning research community to focus on evaluation and validation. Without a rigorous validation of existing algorithms, the methods introduced in the field may be unable to be deployed in practice. In the meantime, this thesis launches an exciting new direction in BIL by proposing a modular framework that combines recent developments in BRL and self-supervised learning. This framework removes the risk of environmental interaction and strengthens robustness by leveraging regularization techniques from the BRL literature. By advancing these two fields in parallel, coupled with rigorous introspection, perhaps we can be closer to AI’s grand goal of large scale safe autonomy.
A.1 Mujoco

Mujoco is a collection of high-dimensional locomotion physical control tasks. The goals for all the tasks is to learn to move forward (e.g. walking, hopping) as fast as possible. The challenges in these tasks come from the high-dimensional state space as well as the agent’s sensitivity to stochasticity in physics. The state space of the agents includes their generalized positions and vehicles, and the action space consists of the torques applied to their joints. The most commonly used and benchmark environments in RL and IL research are (1) Reacher, (2) Hopper, (3) HalfCheetah, (4) Walker, (5) Ant, and (6) Humanoid. These tasks are listed in increasing order of state-space dimension and difficulty. Some of the tasks have fixed episode length, while others have varying durations. In the tasks that have varying episode lengths, early termination usually results from the agent “touching” the ground and hence failing in achieving the intended goal. The table below provides a specification of the tasks. We also include an illustration of three of the tasks for visualization. In our work, we primarily use the Hopper, Walker-2d, and Humanoid environments because we find them to be easier to train.

<table>
<thead>
<tr>
<th>Environment</th>
<th>State Space</th>
<th>Action Space</th>
<th>Fixed Episode Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reacher-v2</td>
<td>11 (continuous)</td>
<td>2 (continuous)</td>
<td>False</td>
</tr>
<tr>
<td>Hopper-v2</td>
<td>11 (continuous)</td>
<td>3 (continuous)</td>
<td>False</td>
</tr>
<tr>
<td>HalfCheetah-v2</td>
<td>17 (continuous)</td>
<td>6 (continuous)</td>
<td>True</td>
</tr>
<tr>
<td>Walker-v2</td>
<td>17 (continuous)</td>
<td>6 (continuous)</td>
<td>False</td>
</tr>
<tr>
<td>Ant-v2</td>
<td>111 (continuous)</td>
<td>8 (continuous)</td>
<td>True</td>
</tr>
<tr>
<td>Humanoid-v2</td>
<td>376 (continuous)</td>
<td>17 (continuous)</td>
<td>False</td>
</tr>
</tbody>
</table>
Figure A.1.1: Illustration of Hopper, Walker, and HalfCheetah tasks. Figure courtesy of [Duan et al., 2016].
B.1 Mujoco Environments

In the Mujoco environments, the expert policy is trained using the benchmark model-free RL algorithm PPO [Schulman et al., 2017] with generalized advantage estimation (GAE) [Schulman et al., 2015b] as the control variate in the policy gradient. The expert is trained for 500 iterations with the following set of parameters for all environments. The trained experts are then evaluated by computing the average over 50 independent trajectories. The training performance curves are included next page.

<table>
<thead>
<tr>
<th>Table B.1.1: Mujoco PPO Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>Actor architecture</td>
</tr>
<tr>
<td>Actor optimizer</td>
</tr>
<tr>
<td>Critic architecture</td>
</tr>
<tr>
<td>Critic optimizer</td>
</tr>
<tr>
<td>Clipping ( \varepsilon )</td>
</tr>
<tr>
<td>GAE ( \gamma )</td>
</tr>
<tr>
<td>GAE ( \tau )</td>
</tr>
<tr>
<td>Training iterations</td>
</tr>
<tr>
<td>Training batch size</td>
</tr>
<tr>
<td>Update epochs</td>
</tr>
<tr>
<td>Update batch size</td>
</tr>
</tbody>
</table>
Table B.1.2: Mujoco PPO Expert Performance

<table>
<thead>
<tr>
<th>Environment</th>
<th>State Space</th>
<th>Action Space</th>
<th>Expert Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reacher-v2</td>
<td>11 (continuous)</td>
<td>2 (continuous)</td>
<td>-8.44 ± 2.75</td>
</tr>
<tr>
<td>Hopper-v2</td>
<td>11 (continuous)</td>
<td>3 (continuous)</td>
<td>1148.32 ± 338.82</td>
</tr>
<tr>
<td>HalfCheetah-v2</td>
<td>17 (continuous)</td>
<td>6 (continuous)</td>
<td>5209.34 ± 109.28</td>
</tr>
<tr>
<td>Walker-v2</td>
<td>17 (continuous)</td>
<td>6 (continuous)</td>
<td>4095.73 ± 1756.84</td>
</tr>
<tr>
<td>Ant-v2</td>
<td>111 (continuous)</td>
<td>8 (continuous)</td>
<td>4004.04 ± 1337.06</td>
</tr>
<tr>
<td>Humanoid-v2</td>
<td>376 (continuous)</td>
<td>17 (continuous)</td>
<td>480.43 ± 100.42</td>
</tr>
</tbody>
</table>
B.2 Expert Trajectories

100 expert trajectories for each environment are generated according to the procedure described in chapter 4. For visualization, we plot their cumulative rewards and durations from highest to lowest.
Figure B.2.1: Mujoco Expert Trajectory Performance Distribution
C

Imitation Learning Models Implementation Details

C.1 Hardware Specification

All models are trained on either a Macbook Pro with a 2.7 GHz Intel Core i7 and 16GB memory, or an Amazon Web Service (AWS) r5.x4large machine with 32 cores, or an AWS GPU p3.2xlarge instance. Most models took about 1-5 hours of clock time to compute on these machines.

C.2 Behavior Cloning (Mujoco)

We use the same BC architecture for all Mujoco tasks experiments in this thesis. The classifier is implemented as a 2-layer feed-forward neural network. We train the model using cross-entropy loss with Adam [Kingma and Ba, 2014] optimizer. The full specification is listed in the table below.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network architecture</td>
<td>RELU $\circ$ Linear(state_dimension,400)</td>
</tr>
<tr>
<td></td>
<td>RELU $\circ$ Linear(300, action_dimension)</td>
</tr>
<tr>
<td>Actor optimizer</td>
<td>Adam w/ rate learning 0.0001</td>
</tr>
<tr>
<td>Training iterations</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Training batch size</td>
<td>full batch (chapter 4) or 100 (chapter 6)</td>
</tr>
</tbody>
</table>

C.3 GAIL

We implement the same GAIL architecture for all Mujoco tasks experiments in this thesis. Our model is implemented using PyTorch. The discriminator is a feed forward neural network with 2 hidden layers of
size 100. Tanh is used as the activation function between layers. The policy network follows the same architecture as the PPO network (B.1.1) used to train the expert policy and the policy update also follows the PPO algorithm.

**Table C.3.1:** Mujoco GAIL Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
</table>
| Discriminator architecture | $\text{Tanh} \circ \text{Linear}(\text{input
dimension},100)$ $\text{Tanh} \circ \text{Linear}(100,1)$ |
| Discriminator optimizer   | Adam w/ rate learning $0.0003$                    |
| Actor architecture        | $\text{Tanh} \circ \text{Linear}(\text{input
dimension},100)$ $\text{Tanh} \circ \text{Linear}(100,\text{action
dimension})$ |
| Actor optimizer           | Adam w/ rate learning $0.0003$                    |
| Critic architecture       | $\text{Tanh} \circ \text{Linear}(\text{input
dimension},100)$ $\text{Tanh} \circ \text{Linear}(100,1)$ |
| Critic optimizer          | Adam w/ rate learning $0.0003$                    |
| Clipping $\epsilon$      | $0.2$                                            |
| GAE $\gamma$             | $0.99$                                           |
| GAE $\tau$               | $0.95$                                           |
| Training steps            | $10^6$                                           |
| Training batch size       | $2048$                                           |
| Update epochs             | $10$                                             |
| Update batch size         | $64$                                             |
### Table C.4.1: Mujoco (DR)BCQ Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-Network architecture</td>
<td>ReLU $\circ$ Linear($input_dimension$,400)</td>
</tr>
<tr>
<td></td>
<td>ReLU $\circ$ Linear(400,300)</td>
</tr>
<tr>
<td></td>
<td>Linear(300, $output_dimension$)</td>
</tr>
<tr>
<td>Q-Network optimizer</td>
<td>Adam w/ rate learning 0.0001</td>
</tr>
<tr>
<td>Actor architecture</td>
<td>ReLU $\circ$ Linear($input_dimension$,400)</td>
</tr>
<tr>
<td></td>
<td>ReLU $\circ$ Linear(400,300)</td>
</tr>
<tr>
<td></td>
<td>Linear(300, $output_dimension$)</td>
</tr>
<tr>
<td>Actor optimizer</td>
<td>Adam w/ rate learning 0.0001</td>
</tr>
<tr>
<td>VAE encoder architecture</td>
<td>ReLU $\circ$ Linear(750,750)</td>
</tr>
<tr>
<td></td>
<td>ReLU $\circ$ Linear(750,750)</td>
</tr>
<tr>
<td>VAE decoder architecture</td>
<td>ReLU $\circ$ Linear($input_dimension$ + $output_dimension$,750)</td>
</tr>
<tr>
<td></td>
<td>ReLU $\circ$ Linear(750,750)</td>
</tr>
<tr>
<td></td>
<td>Linear(750,$output_dimension$)</td>
</tr>
<tr>
<td>$\tau$</td>
<td>0.005</td>
</tr>
<tr>
<td>discount rate</td>
<td>0.99</td>
</tr>
<tr>
<td>Training iterations</td>
<td>20000</td>
</tr>
<tr>
<td>Training batch size</td>
<td>100</td>
</tr>
</tbody>
</table>
The evaluation results for BC (batch size 100) and DRBCQ Hopper-v2 and Humanoid-v2 environments are displayed here. Best Performance, Running Performance, and Noisy Actions ($\epsilon = 0$) are included as these are the most informative metrics from our studies in chapter 6. For compactness and ease of comparison, we jointly present the two models’ results side by side. The main conclusion is that Hopper and Humanoid environments are different from Walker2d in that models trained with mixed trajectories do not perform visibly worse than models trained with pure expert trajectories - though the degraded trajectories themselves are qualitative worse (e.g. much lower cumulative rewards in their respective environments). As such, DRBCQ and BC’s results on these environments are not entirely consistently with our conclusions in chapter 6. Due to time constraint, we leave it to future work to investigate why this is the case. Investigating this phenomenon is an interesting direction, and would add to a growing literature that understands how some unhidden features of RL environments may contribute to the observable performance differences of RL algorithms [Pardo et al., 2017] [Kostrikov et al., 2019].
D.1 Hopper

D.1.1 Best Performance

![BC Hopper Best Performance](image1)

![DRBCQ Hopper Best Performance](image2)

**Figure D.1.1:** BC & DRBCQ Hopper Best Performances

D.1.2 Running Performance

![BC Hopper Running Performance](image3)

![DRBCQ Hopper Running Performance](image4)

**Figure D.1.2:** BC & DRBCQ Hopper Running Performances

D.1.3 Noisy Actions

![BC Hopper Noisy Actions](image5)

![DRBCQ Hopper Noisy Actions](image6)

**Figure D.1.3:** BC & DRBCQ Hopper Noisy Actions
D.2 Humanoid

D.2.1 Best Performance

Figure D.2.1: BC & DRBCQ Humanoid Best Performances

D.2.2 Running Performance

Figure D.2.2: BC & DRBCQ Humanoid Running Performances

D.2.3 Noisy Actions

Figure D.2.3: BC & DRBCQ Humanoid Noisy Actions, $\epsilon = 0$
E

Missing Background

E.1 Proximal Policy Optimization (PPO)

PPO [Schulman et al., 2017] is a state-of-art model-free policy gradient (PG) algorithm. It optimizes a surrogate PG objective function using stochastic gradient descent. Recognizing that the original PG objective

\[ L_{\text{PG}}(\theta) = \mathbb{E}_t \left[ \log \pi_\theta(a_t|s_t)A_t \right] \]

is unstable due to high variance and sample inefficiency, and that the TRPO [Schulman et al., 2015a] policy update

\[
\text{maximize}_{\theta} \quad \mathbb{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A_t \right] \\
\text{subject to} \quad \mathbb{E}_t [KL(\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t))] \leq \delta
\]

is difficult to implement and optimize, Schulman et al. [2017] proposes a simple alternative objective

\[ L_{\text{PPO}}(\theta) = \mathbb{E}_t \left[ \min (r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t) \right] \]

that clips the gradient to a constrained region to prevent overshooting. The resulting algorithm, Proximal Policy Optimization (PPO), is found to be more stable and requires fewer samples. All expert models in this thesis are trained using PPO. See B for more detail.

E.2 Variational Auto-Encoder (VAE)

The excellent short introduction to VAE [Kingma and Welling, 2013] in [Fujimoto et al., 2018a] is reproduced here. A variational auto-encoder (VAE) is a generative model which aims to maximize the marginal log-likelihood \( \log p(X) = \sum_{i=1}^{N} \log p(x_i) \) where \( X = \{x_1, \ldots, x_N\} \). Each datapoint \( x_i \) is drawn
from a conditional distribution \( x_i \sim p(x_i | z_i) \) given the latent variable \( z_i \sim p(z) \). Maximizing the marginal likelihood \( \log p(X) = \log \sum_{i=1}^{N} \int p(x_i | z_i) p(z_i) dz \) is intractable. Instead, VAE parameterizes the conditional distribution \( p_\phi(X|z) \) by a neural net \( \phi \), defines a posterior \( q_\theta(z|X) = \mathcal{N}(z|\mu(X), \sigma^2(X)) \), parameterized by a neural net \( \theta \), and jointly optimizes the variational lower-bound:

\[
\log p(X) \geq \mathbb{E}_{q_\theta(X|z)}[\log p_\phi(X|z)] + D_{KL}(q_\theta(z|X)||p(z))
\]

\( q_\theta(z|X) \) is also known as the **encoder**, and \( p_\phi(X|z) \) the **decoder**. To generate realistic samples \( x \) at inference time, \( z \sim p(z) \) is drawn from the prior and then passed through the decoder \( p_\phi(\cdot|z) \).
References


