Inverse Reinforcement Learning Under the Batch Constraint (Draft)

A DISSERTATION PRESENTED
BY
DONGHUN LEE
TO
THE INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ENGINEERING
IN THE SUBJECT OF
COMPUTATIONAL SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS

MAY 2019

Thesis advisor: Finale Doshi-velez Donghun Lee

Inverse Reinforcement Learning Under the Batch Constraint (Draft)

Abstract

This thesis is an attempt to understand Inverse Reinforcement Learning (IRL) in a special setting: an agent is given a fixed batch of expert demonstrations and must find a reward function that best explains the expert demonstrations without the interaction with an environment or the knowledge of dynamics. The setting which we term *the batch constraint* is common in such domains as health care, or finance where data collection is severely restricted. Most IRL algorithms are not applicable in the setting, for they assume the existence of a simulator or the knowledge of environment dynamics. We attempt to fill in the gap.

This thesis mainly consists of four parts. In the first part, we motivate the standard formulation of IRL and explain why IRL is generally difficult. In the second part, we review existing batch IRL methods and analyze how the batch constraint makes IRL more difficult. In the third part, we introduce a novel batch IRL method, the main contribution of the thesis. The method efficiently addresses most challenges intrinsic to IRL as well as the challenge uniquely posed by the batch constraint. We provide empirical evaluations of the method on classical control tasks and a real-world clinical task. Lastly, we discuss an alternative kernel-based approach that was recently proposed. To our knowledge, we provide the first empirical analysis that addresses questions untapped in the original work.

Contents

Listing of figures

List of Tables

To my family.

Acknowledgments

This thesis came about in a very close collaboration with my wonderful advisor, Finale Doshi-velez. She is an exemplar leader who has continuously guided me in the right direction and enlightened me to do the best work I could. Without her wisdom, this thesis would not have become a reality. I thank her with the utmost sincerity.

I would also like to thank Srivatsan Srinivasan, my brilliant colleague. The major contributions of the thesis have been produced by our joint work that spanned for longer than a year. Whenever we were stuck and I felt my skills were just inadequate, he always convinced me otherwise with his magical optimism and we would find practical solutions. I also feel very grateful to my colleagues at the RL group: Omer Gottesman, Jiayu Yao, Arjumand Masood, and Joe Futoma. I thank them for all the thought-provoking conversations.

I take my hat off to Wael Alghamdi and Oussama Dhifallah whom showed me the mythical ideal researchers do exist. Wael taught me the importance of fundamental thinking and Oussama the importance of perseverance. It has been a great joy to work with them. I also thank my colleagues at SEAS including Camilo Fosco, Justin Lee, Vincent Cas, Paul Blankley, Ryan Janssen, and Ismail Atitallah.

I cannot express my gratitude enough to my mentors at IACS, Pavlos Protopapas, Weiwei Pan, and Daniel Weinstock who have supported me with great knowledge and wisdom during my study. I dream of paying back or forward

for our community one day.

Lastly and foremost, this thesis is dedicated to my family whom I feel eternally indebted to. My mother has always believed in me, with unconditional passion. She taught me the importance of continuously educating myself. My father, peacefully in heaven, inspired to become a researcher and taught me the joy of learning. My grandparents supported me with love far beyond what I deserve. I thank my wife for her incredible kindness and encouragement. Thanks to her, I could overcome my day-to-day graduate school struggles. I thank my family again for always making me want to be a better version of myself.

Find a good teacher that will keep the game fun.

Paula Creamer

1 Introduction

In most control tasks, the goal is to find an optimal policy, a decision rule one can trust to perform a given task optimally. Reinforcement Learning (RL) is an indirect formalism that derives the optimal policy. Rather than prescribing what to do directly, RL uses a reward function, as a proxy, to evaluate how good or bad situations are and in turn, prescribes what to do. The implicit assumption is that reward is the the most succinct and transferable definition of a task [1].

However, it is frequently challenging to manually specify a reward function [38]. Consider the task of designing a self-driving algorithm that is expected to be deployed to various places around the world. Even the most highly-experienced domain expert would not be able to fully specify how a reward function should be composed, for there would be too many different situations to consider.

What may be easier is to ask the expert to demonstrate what the right actions are. After all, they are expert in the virtue of knowing what to do given different

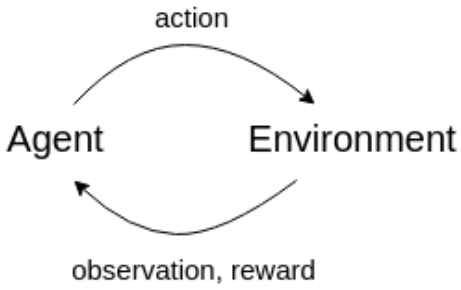

Figure 1.1.1: Overview of Reinforcement Learning

situations. As for the self-driving example, one could ask human drivers to drive a car in various terrains with sensors attached to the car record the images and the actions. Of course, the task is not as simple as it sounds. The agent may not have expert demonstrations for all situations it may encounter. This implies the agent must *infer* what the expert would do for unseen situations. Furthermore, even with expert demonstrations, designing an algorithm that can drive a car is complex. Unlike common supervised learning tasks such as image classification, driving a car is not a single decision task: it is a sequence of decisions over time. What the agent will see in the future may change due to the decisions it had already made in the past. This feedback loop is one of the key challenges that characterizes reinforcement learning that we will introduce in this chapter.

1.1 Reinforcement Learning

Reinforcement learning is a general framework designed to tackle solving *sequential decision making* problems. Sequential decision making problems are typically concerned with an agent situated in an environment. The agent aims to behave optimally for an array of states. The agent interacts with the environment by taking actions and receives observations and rewards from the environment for each action it takes. The agent seeks to learn an optimal *policy* by maximizing total rewards under uncertainty about the environment through a trial-and-error approach.

1.2 Learning from Demonstration

In solving a reinforcement learning problem, an agent does not have to start its training from scratch. Instead, it can ask an expert for demonstrations before it begins to learn [49]. i.e. an expert demonstrates optimal examples for a subset of state-action pairs and from the fixed set of examples, an agent seeks to learn an optimal policy, a mapping from states to actions. Such an approach is called Learning from Demonstration (LfD). The two major approaches to LfD are as follow.

Imitation via Policy: This approach is known as *Imitation Learning* (IL) or *behavior cloning*. The goal is to directly learn an optimal policy from the examples provided by an expert. The problem can be formulated as a multi-class supervised learning problem. Despite being conceptually simple, it is known that small errors in a learned policy can accumulate over time, potentially leading to situations that were not encountered during training [47].

Imitation via Reward Function: This approach is known as *Inverse Reinforcement Learning*. The main assumption is that an expert behaves optimally with respect to some unknown reward function. The goal is to recover a reward function under which an expert is assumed to operate and subsequently, optimize the reward function to learn an optimal policy. The main motivation is that recovering a reward function is the most succinct, robust, and transferable representation of a task [1].

The reinforcement learning problem formulation implicitly assumes there exists a known, fixed reward function. The reward function essentially transfers knowledge of how to perform a given task optimally. For many real-world tasks, such reward functions are unknown. Hence, one must engineer an appropriate reward function that corresponds to the optimal behavior. However, manually engineering a reward function is often not trivial [37].

1.3 Batch Learning Problem

The general reinforcement learning problem assumes an agent interacting with an environment and encountering a sequence of transitions $\{(s_t, a_t, r_t, s_{t+1})\}$. On the contrary, in the batch setting, we assume the agent is not allowed to interact with the environment. In the general case, an agent collects training samples from scratch (exploration) and aims to learn the optimal behavior from the samples (exploitation).

The batch constraint, simply put, means no simulator/environment interactions *and* no knowledge of environment dynamics. Instead, an agent is given a set of training samples fixed a priori and must learn the best possible behavior with the given data alone [8]. Considering the batch setting makes sense for such problem domains as health-care, finance, or education where data acquisition is highly limited for cost or regulation reasons. An implicit assumption is that exploration has been done already. Batch algorithms, as opposed to online algorithms, are often more sample efficient and stable [31].

1.4 Representation Learning

Representation Learning is a class of algorithms that aim to automatically discover a feature map that is expected be useful for some predefined task. Inverse Reinforcement Learning is known to be sensitive to the choice of a feature map and it is not easy to manually specify a feature map. Hence, attempts have been made to learn appropriate feature maps from data [22, 32].

1.5 Deep Learning

Deep learning refers to a learning algorithm based on a deep neural network. It usually entails a loss function that is expected to approximate a target function, a neural network architecture that consists of parameters to optimize, and an optimization method such as stochastic gradient descent. While relatively weak

on giving theoretical guarantees, neural networks have been shown to be capable of performing well on difficult control tasks. [35] nonlinear models lack convergence guarantees.

1.6 Contributions

We summarize the main contributions of the thesis as follow.

- In Section 3.3, we specify the reason the batch setting makes IRL challenging and explore the ways to address the challenge.
- In Section 3.4, we provide a survey of batch IRL methods.
- In Chapter 4, we propose a novel framework that altogether addresses practical challenges in batch IRL.
- In Section 4.3.1, we propose DSFN, a novel batch inverse reinforcement learning method that estimates feature expectations, the key quantity in performing IRL.
- In Section 4.4, we propose TRIL, a novel regularization scheme for imitation learning that significantly improves the performance of a policy.
- In Chapter 5.1, we provide an empirical evaluation of a novel kernel-based Imitation Learning method, as an alternative approach.

It is the supreme art of the teacher to awaken joy in creative expression and knowledge.

Albert Einstein

2 Background

In this chapter, we establish a formalism for IRL. Furthermore, we discuss fundamental challenges intrinsic to IRL and evaluate major approaches. The challenges of IRL we enumerate in this chapter motivate the design of the novel method we will introduce in Chapter 4.

2.1 Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical model for discrete-time sequential decision process. In an MDP, an agent takes action a_t at time t in a state s_t. Subsequently, the agent receives a reward r_t and a next state s_{t+1}. And the stochastic process continues until the agent reaches a terminal or absorbing state denoted by s_T. The key modelling assumption in an MDP is the *Markovian* assumption. That is, the transition probability distribution of the next state

depends only on the current state-action pair: $\mathbb{P}(s_{t+1}|h_t) = \mathbb{P}(s_{t+1}|s_t, a_t), \forall t \in \mathbb{N}$ where h_t denote a sequence $\{(s_0, a_0), (s_1, a_1), \ldots, (s_t, a_t)\}$, also called an episode.

2.1.1 Finite-horizon, Stationary MDP

We establish the notations necessary to describe a finite-horizon, stationary MDP.

- $\mathcal{S}$ (state space) is a set of states of the environment.
- $\mathcal{A}$ (action space) is a set of actions of which an agent may choose at each time step t.
- T (transition probability distribution) represents $T \triangleq \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a)$, the probability of transitioning to s_{t+1} given s_t and a_t. T is sometimes called the (model) dynamics.
- T_0 (initial state distribution) represents a marginal probability distribution over a set of initial states $T_0 \triangleq \mathbb{P}(s_0)$, where $\mathcal{S}_0 \subset \mathcal{S}$.
- R (reward function) is a deterministic map $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}_{\geq 0}$. Depending on contexts, reward function can also be defined as $R(s)$ or $R(s, a, s')$. We define $r_t = R(s_t, a_t)$.
- γ (discount factor) discounts future rewards at a fixed rate of $\gamma \in [0, 1)$.

Though an MDP is a general framework, we will focus, in our work, on a specific instance of MDP: finite-horizon, stationary MDP. First, the finite-horizon assumption is when the number of time steps an agent takes to reach a terminal or an absorbing state is finite, $|h_T| < \infty$. Second, the stationary assumption means T does not change over time. That is:

$$\mathbb{P}(S_{t'+1} = s'|S_{t'} = s, A_{t'} = a) = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a), \forall t, t' \in \mathbb{N}$$

Lastly, $\mathcal{S}$ or $\mathcal{A}$ can be finite or infinite. In our work, we will mostly focus on ant infinite state and finite action MDP.

2.1.2 Policy and Value Function

An agent is situated in an MDP and its goal of is to learn an optimal policy which we will define later in this section. As a preliminary, we establish a few notations. Let $G_t = \sum_{t=0}^{T} \gamma^t r_t$ denote the discounted total rewards. Let $\pi(a|s)$ denote **policy**, a probability distribution over $\mathcal{A}$ conditioned on $\mathcal{S}$. If policy is deterministic, we use $\pi(s)$. We define **state-value function** as the expected total reward starting at s_t following $\pi(a|s)$ onward.

$$V^{\pi}(s_{t'}) = \mathbb{E}_{\pi}\big[\sum_{t=t'}^{T} \gamma^t r_t | S_t = s_{t'}\big]$$

We define **action-value function** as the expected total reward starting by taking a_t at s_t and following $\pi(a|s)$ onward.

$$Q^{\pi}(s_{t'}, a_{t'}) = r_t + \mathbb{E}_{\pi}\big[\sum_{t=t'}^{T} \gamma^t r_t | S_t = s_{t'}, A_t = a_{t'}\big]$$

Here, $\mathbb{E}_{\pi}$ is taken over some visitation distribution over $\mathcal{S} \times \mathcal{A}$ induced by $\pi(a|s)$, $\mathbb{P}(s'|s,a)$ and $\mathbb{P}(s_0)$. Concretely, it means $s_{t+1} \sim \mathbb{P}(s_{t+1}|s_t)$, $a_t \sim \pi(a_t|s_t)$ and $s_0 \sim \mathbb{P}(s_0)$ if $t' = 0$. We define optimal policy:

$$\pi^*(a|s) \stackrel{\Delta}{=} \arg\max_{\pi \in \Pi} V^{\pi}(s), \forall s \in \mathcal{S}$$

where Π is the set of all polcies. We define **Bellman equation**:

$$V^{\pi} = \mathbb{E}_{\pi}\big[R(s,a) + \gamma V^{\pi}(a'|s')\big]$$

We also define **Bellman Optimality equation**:

$$V^{\pi}(s) = \max_{a \in \mathcal{A}} \mathbb{E}_{\pi}\big[R(s,a) + \gamma V^{\pi}(s')\big]$$

2.2 Inverse Reinforcement Learning

Inverse Reinforcement Learning is a process of learning a reward function from expert demonstration such that the expert behavior is recovered by optimizing the reward function. Let π_e denote the expert policy, being expert in the virtue of $\arg\max_a \pi_e(a|s) = \arg\max_a \pi^*(a|s), \forall s \in \mathcal{S}$. We assume that an agent is given a fixed set of demonstrated trajectories—a collection of finite-horizon episodes $\mathcal{D}_\pi = \{h_T\}, |D_\pi| = N \in \mathbb{N}$ where an episode h_T is a finite sequence $s_0, a_0, s_1, a_1, \ldots, s_T$ following π. We assume each trajectory is fully observable.

Definition 1. *(The IRL Problem): Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}_e, T, T_0, \gamma)$ be a finite-horizon, stationary MDP and π_e an optimal policy for $\mathcal{M}$. Let $\mathcal{M} \setminus \mathcal{R}_e$ denote an $\mathcal{R}_e$-absent MDP. Let $\mathcal{D}_e$ denote a collection of demonstrated trajectories sampled by π_e. Given $\mathcal{M} \setminus \mathcal{R}_e$ and $\mathcal{D}_e$, find $\widetilde{\mathcal{R}}$ such that solving $\mathcal{M}$ equipped with $\widetilde{\mathcal{R}}$ recovers π_e. Note that we do not assume an agent has the direct knowledge of π_e or the environment dynamics T, T_0.*

2.2.1 Ambiguities of IRL

The IRL problem is ill-defined due to the two fundamental ambiguities [37], as can be seen in Figure 2.2.1. The first ambiguity is that there are many reward functions that can produce an optimal policy. If there exists a solution to the IRL problem, it implies there are infinitely many solutions. This follows from the fact that the optimal policy is invariant under affine transformations of a feasible reward function. The second ambiguity is that there are many optimal policies that can produce an empirical distribution of trajectories as demonstrated by the expert policy. Therefore, it is necessary for an IRL algorithm to have a means to resolve the ambiguities. For instance, one popular remedy is to use a maximum entropy objective [59] to resolve the second ambiguity. To address the first ambiguity, a recent work [11] aims to learn a reward function that does not vary when environment dynamics (transition model) change.

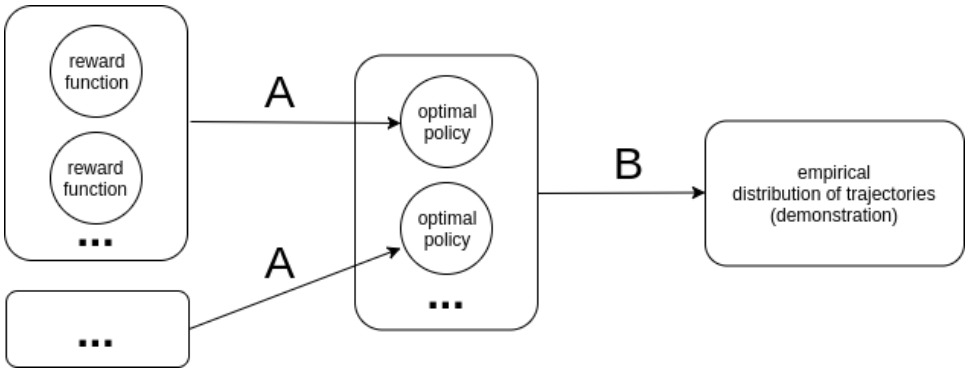

Figure 2.2.1: Intrinsic Ambiguities of IRL: IRL is an ill-posed problem. There are two fundamental ambiguities to resolve. The first ambiguity (A) is that there are many reward functions that can produce the same optimal policy. The second ambiguity (B) is that there are many optimal policies that can produce the same empirical distribution of trajectories. To resolve the ambiguities, one must introduce additional constraints.

2.2.2 Defining the Closeness to Expert Behavior

In the IRL problem, the aim is to imitate expert behavior. For all practical matters, an agent hopes to imitate expert behavior so it can perform some pre-defined task optimally. Hence, it makes sense to claim that the ultimate aim of the IRL problem is to get the agent's performance, however measured, to be on par with the expert's performance.

With this in mind, we define a metric that measures the closeness to the expert behavior is by comparing the values of expert and agent policy. It follows from the definition of π_e that $V_e^{\pi}(s_0) = V^*$, the optimal value function. We measure the closeness to the expert behavior in the virtue of how close the value of an agent's policy is to the value of the expert policy.

Definition 2. *Value Error (VE): the difference in the value of the two policies* $VE \triangleq ||\mathbb{E}_{s_0}[V^{\pi_e}(s_0)] - \mathbb{E}_{s_0}[V^{\tilde{\pi}}(s_0)||$.

Definition 3. *Empirical Value Error (EVE): the difference in the value of the two policies given the empirical expert demonstration* $\widehat{VE} \triangleq ||\frac{1}{|\mathcal{D}_e|}\Sigma_{s\in\mathcal{D}_e}V^{\pi_e}(s)] - V^{\tilde{\pi}}(s)||$

2.2.3 Expert Reward as a Closeness Metric

As a digression, we can consider other metrics to define the closeness to expert behavior. For instance, we could consider using expert reward function $\mathcal{R}_e$ that is generally not available to an agent. Let us assume there exists an oracle that knows $\mathcal{R}_e$ and computes the following loss function $||\mathcal{R}_e - \widetilde{\mathcal{R}}||$ where each reward function is vectorized on some finite number of *s*-*a* pairs. That is, an agent chooses some finite subset of the state-action space and define a reward function, $\widetilde{\mathcal{R}}$, as it sees best fit. The agent receives a loss for its suggestions and optimize $\widetilde{\mathcal{R}}$. It may be tempting to think the loss function is sufficient to perform a successful inference. Such a setting, however, is not desirable for two reasons. First, two reward functions of a small difference can produce different policies. Intuitively, if the two reward functions disagree on some small subset of the state-action space and if the subset has a big impact on what can happen subsequently (e.g. falling off a cliff), the policies that will be induced by the two reward functions can be drastically different. Second, two reward functions of a large difference are not necessarily bad. Hence, the loss function would ignore other feasible solutions.

2.2.4 Occupancy Measure Matching

As a digression, we consider an alternative metric to define the closeness to expert behavior: occupancy measure which we will define soon. The big idea is that for every occupancy measure, there is a policy uniquely associated with it and hence, when two occupancy measures coincide, the two associated policies also coincide. In this section, we follow the notations [19] and the details of the proof of the key arguments can be found in the paper. We define occupancy measure: $\rho_\pi : S \times A \to \mathbb{R}$.

Definition 4. *State Occupancy Measure:* $\rho_\pi(s) \stackrel{\Delta}{=} \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(S_t = s)]$.

Definition 5. *State-action Occupancy Measure:* $\rho_\pi(s, a) \stackrel{\Delta}{=} \pi(a|s)\rho_\pi(s)$.

Notice $\rho_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(S_t = s)] = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_{s_0,\pi}(S_t = s)$ where $\mathbb{I}$ is an indicator function and $\mathbb{P}_{s_0,\pi}(S_t = s)$ refers to the probability of encountering s at

time step t, following π and starting at $s_0 \sim \mathbb{P}(s_0)$. It is γ-discounted state visitation distribution of a policy. An intuitive view is how often an agent encounters certain state-action pairs when interacting with the environment following π.

Using occupancy measure, we can write $\mathbb{E}_\pi[R(s,a)] = \sum_{s,a} R(s,a)\rho_\pi(s,a)$. Let Π the set of stationary policies. Let $\mathcal{O} \stackrel{\Delta}{=} \{\rho_\pi \mid \pi \in \Pi\}$. It is known that there is a bijection between Π and $\mathcal{O}$ [43]. It follows that for every ρ there exists a unique π associated with it where π can be identified to be $\pi = \frac{\rho(s,a)}{\sum_{a'} \rho(s,a')}$. Hence, matching occupancy measures implies matching policies.

As a final note, we remark that we will revisit this concept in the chapter on kernel-based imitation learning. There we will see the connections between occupancy measures and policy matching.

2.2.5 Solving the IRL Problem

Let us define two primitive operators to understand the general procedure of solving the IRL problem. Let *IRL* be an operator $IRL : \Pi \mapsto \mathcal{R}$:

$$IRL(\pi_e) = \arg \max_{||\varphi|| \leq 1} \mathbb{E}_{\pi_E}[R(s,a)] - \mathbb{E}_{\tilde{\pi}}[R(s,a)] \tag{2.1}$$

Let *RL* be an operator $RL : \mathcal{R} \mapsto \Pi$ (we omit other elements of $\mathcal{M}$ to make our argument clear):

$$RL(R) = \arg \max_{\tilde{\pi} \in \Pi} \mathbb{E}_{\tilde{\pi}} \left[\sum_{t=0}^{T} \gamma^t R(s,a) \right] \tag{2.2}$$

Then, solving the IRL problem is a loop where each iteration we compute $RL \circ IRL$ until the empirical value error is smaller than $\varepsilon > 0$.

It takes some explanation for the line 4 in the algorithm. The IRL solving procedure searches for a reward function that assigns high reward to the expert policy and low reward to the candidate policy ($IRL(\pi_e)$). Subsequently, the IRL solving procedure searches for a policy that minimizes the expected total return

Algorithm 1 A General Procedure of Solving the IRL problem

Input: D_e (Demonstration)
Parameter: w
Output: R_w

1: Initialize R_w in a parameterized form (linear, non-linear or a distribution over $\mathcal{R}$)
2: Solve $RL(R_w)$ and let the solution be $\widetilde{\pi}$ (i.e. solve $\mathcal{M} \setminus \mathcal{R}$ under R_w)
3: Measure the closeness of $\widetilde{\pi}$'s behavior to the expert's demonstration (D_e)
4: Solve $IRL(\widetilde{\pi})$ and let the solution be R_w.
5: Repeat 2-4 until the two behaviors are close enough.
6: **return** R_w

$(RL(R_w))$.

2.2.6 Challenges of IRL

Given the outline of solving the IRL problem in 1, we introduce some of the common challenges in IRL. The first challenge is a computational cost. We claimed that IRL is a loop (line t) where at each iteration a reinforcement learning problem must be solved (the line 2). If the computational cost of solving RL is $O(M)$ and the length of the loop is $O(N)$, the overall cost is $\Omega(M^N)$. This is the largest obstacle generally considered in IRL approaches. For a fairly large-scale problem, unless necessary modifications are made, 1 may not be tractable.

The second challenge is the choice of reward hypotheses space (line 1). By reward hypotheses space, we mean a set of reward functions that can be represented by our modelling choice (e.g. linear or non-linear). If a task is relatively complex, a reward function typically needs to distinguish various subtle behaviors. In many IRL literatures, reward function is assumed to be linear in its parameters. To enhance the richness of the reward hypotheses space, also a feature map $\varphi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^d$ is defined. Unless sufficient domain knowledge is available, the choice of a feature map may have a big impact on the success of solving the IRL problem. Hence, a few approaches have been proposed to reduce the sensitivity to feature maps [33]. Reward function motivates an agent towards

a certain behavior, by specifying relative preference over $\mathcal{S} \times \mathcal{A}$. A typical modelling assumption is the linear reward assumption. Let the linear reward function $R(s,a) \triangleq w^\top \varphi(s,a)$, the reward parameters $w \in \mathbb{R}^d$, and the feature map $\varphi(s,a) \in \mathbb{R}^d$. Let R_{linear} be the hypotheses space of linear reward functions. The linear reward assumption may be too restrictive, if a target behavior—that will be implicitly intended by some reward function—is complex. To obtain a richer hypotheses space of reward functions, one may consider a non-linear reward function represented by a neural network [10]. One possible drawback of using non-linear reward function is that it may be difficult to interpret the reward function. A linear reward function, on the other hand, is relatively more interpretable and hence, may help better analyze the behavior encouraged by the reward function. To make a more rich hypotheses space, we may consider a feature map $\varphi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^d$. Determining what φ to use is problem-dependent and is a hard engineering task to specify φ manually.

The last challenge is the generalizability of an IRL solution. The generalizability or transferability matters most in a multi-task setting: there are multiple goals where an agent is to achieve one goal at some chunk of time and move to another, but an environment dynamics remains fixed. In such a case, one desirable property of reward function models is the decomposition of reward assignment—one attributable to specific goals and to the invariant environment dynamics [11].

2.3 Imitation Learning

With all the challenges of IRL, one may ask why learn a reward function at all. Is there not a simpler approach? This motivates another closely related sub-field of Learning from Demonstration: Imitation Learning (IL). A simpler alternative is called imitation learning (IL) [47] that bypasses the computationally expensive loop. The simplest IL method would arguably be to reduce the main problem to a supervised learning problem. It has been shown the simple reduction is rarely sufficient due to the accumulation of errors. [24] showed a lower bound on the

test loss of a simple supervised IL algorithm is $\tilde{O}(\varepsilon T^2)$. It tells even with a small classification error at each time step $\varepsilon > 0$, the total error will accumulate quadratically in the length of a trajectory T.

Despite the simplicity and low computational costs of IL, one may still consider IRL for different contexts such as the transfer of a task or the interpretation of expert motivation [11].

	IRL	IL
focus	reward	policy
strength	interpretable/generalizable	simple/low computational costs

Table 2.3.1: Inverse Reinforcement Learning vs. Imitation Learning

We must use what we are given, and we must use it the best we can.

Robin McKinley

3

Review of Batch Inverse Reinforcement Learning

3.1 Introduction

In this chapter, we review a few inverse reinforcement learning methods that are either specifically designed to account for the batch constraint or can potentially work under the batch constraint. Most existing IRL methods assumes either the ability to evaluate a policy on a simulator or a real-world environment, or the knowledge of environment dynamics. Since such an assumption is usually tightly coupled with the inner workings of the methods, it is unlikely to expect them to work out of the box under the batch constraint. The story would be that an agent hopes to continuously improve its policy yet it cannot directly monitor the progress of its policy improvement process. The agent must find a way to estimate

the performance of the policy improvement progress, given the batch of data alone. That is precisely the need existing batch inverse reinforcement learning methods cater to.

We will begin with the review of Apprenticeship Learning [1], a foundational IRL method. By design, the method is not suitable for the batch setting. We will discuss why so and in the subsequent chapter, we will show that the method can be adapted to work under the batch constraint. In addition, we will review existing batch IRL methods and conclude with promising alternative attempts that will not be thoroughly discussed in the thesis.

3.2 Review of Apprenticeship Learning

In this section, we review a foundational IRL method called Apprenticeship Learning (AL) [1]. AL was among the first methods to establish a general template for solving the IRL problem by iteratively solving an MDP. In its formulation, an expert is assumed to operate optimally with respect to some reward function. The reward function is assumed to be unknown to an apprentice (agent). The expert provides a set of demonstrations and the apprentice, with the demonstrations alone, must derive a policy whose performance is close enough to the expert policy. The key assumption is that the expert's reward function can be expressed as a linear function with some known basis features. The AL algorithm follows 1 quite faithfully.

3.2.1 Preliminaries

Let $\mathcal{M}$ be an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}_{linear}, T, T_0, \gamma)$. Let $\mathcal{R}_{linear}$ denote the linear reward hypotheses space $\mathcal{R}_{linear} = \left\{R(s,a) \mid R(s,a) = w^\top \varphi(s,a), w \in \mathbb{R}^d, \varphi(s,a) \in \mathbb{R}^d, ||\varphi(s,a)||_2 \leq 1\right\}$. Let the set of (finite-horizon) trajectories demonstrated by expert policy $D_e = \{\tau\}_{i=1:N}$ and $\tau = s_0, a_0, s_1, \ldots, s_T$. For the rest of notations, we follow the notations we establish in the introduction.

3.2.2 Feature Expectations

$$\mathbb{E}_{s_0}\left[V^{\pi}(s_0)\right] = \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T}\gamma^t R(s_0, a_0)\right] \tag{3.1}$$
$$= \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T}\gamma^t w^\top \varphi(s_0, a_0)\right] \tag{3.2}$$
$$= w^\top \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T}\gamma^t \varphi(s_0, a_0)\right] \tag{3.3}$$
$$\tag{3.4}$$

Notice the expectation is taken with respect to a random sequence $s_0 \sim T_0, a \sim \pi, s' \sim T(s'|s, a)$. We define *feature expectation* of a policy:

$$\mu^{\pi} \stackrel{\Delta}{=} \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T}\gamma^t \varphi(s_0, a_0)\right] \tag{3.5}$$

Wherever convenient, we overload the notation by defining *conditional feature expectations* $\mu^{\pi}(s, a) \stackrel{\Delta}{=} \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T}\gamma^t \varphi(s_0, a_0) \mid S_0 = s, A_0 = a\right]$. The feature expectations $\mu(\pi)$ for a state action pair under any policy π is defined as the expected discounted accumulated "feature visitations" induced by π. The feature expectations can be interpreted as the state-action space an agent is expected to encounter following π. Finally, we define *empirical feature expectations* $\widehat{\mu}^{\pi} = \frac{1}{|D|}\sum_{\tau \in D}\sum_{t=0}^{T}\gamma^t \varphi(s_t, a_t)$ where D is a collection of trajectory samples following some policy π. For instance, let $\widehat{\mu}_e$the feature expectation of an expert demonstration D_e.

3.2.3 Maximum Value Difference as Performance Guarantee

Recall the goal of an apprentice is to minimize the value error $VE(\pi_e, \widetilde{\pi}) = \mathbb{E}_{s_0}[V^{\pi_e}(s_0)] - \mathbb{E}_{s_0}[V^{\widetilde{\pi}}(s_0)]$ also known as Maximum Mean Discrepancy (MMD). We want to solve $\min_{\widetilde{R} \in \mathcal{R}_{linear}} VE(\pi_e, \widetilde{\pi})$. However, it is not straightforward to solve the problem, since the value error depends on reward function. Instead, we establish a uniform bound, the maximum difference of the two values. Let *MMD* denote the quantity.

Algorithm 2 Apprenticeship Learning

Input: μ_e, $\mathcal{M}$, ε
Parameter: w
Output: R_w, Π

1: Initialize $\Pi = \emptyset$.
2: Set an initial policy $\pi^{(0)}$ and add it to Π. Estimate $\mu^{\pi^{(0)}}$ via Monte Carlo.
3: Set $t^{(i)} = \|\mu^{\pi^{(0)}} - \mu_e\|_2$. Set $i = 1$
4: **while** $t^{(i)} > \varepsilon$ **do**
5: Compute $t^{(i)} = \max_{w, \|w\|_2 \leq 1} \min_{j \in [i-1]} w^\top (\mu_e - \mu^{\pi^{(j)}})$. Let $w^{(i)}$ be the maximizer.
6: Solve $\mathcal{M} + \mathcal{R}_{w^{(i)}}$. Set the solution $\pi^{(i)}$ and add it to Π.
7: Set $i = i + 1$.
8: **end while**
9: **return** $\mathcal{R}_{w^{(i-1)}}$ and Π

$$MMD(\pi_e, \widetilde{\pi}) = \sup_{||w||_2 \leq 1} \mathbb{E}_{s_0}[V^{\pi_e}(s_0)] - \mathbb{E}_{s_0}[V^{\widetilde{\pi}}(s_0) \tag{3.6}$$

$$= \sup_{||w||_2 \leq 1} w \cdot \left(\mu_e - \mu^{\widetilde{\pi}} \right) \tag{3.7}$$

$$= ||\mu_e - \mu^{\widetilde{\pi}}||_2 \tag{3.8}$$

The last line follows from Cauchy-Schwartz inequality and there exists a closed form solution $w^* = \frac{\mu_e - \mu^{\widetilde{\pi}}}{||\mu_e - \mu^{\widetilde{\pi}}||_2}$ (the maximizer, assuming a maximum exists). Moreover, equation (3.8) implies the value error (MMD) is upper-bounded by $\|\mu_e - \mu^{\widetilde{\pi}}\|_2$. Hence, we have a sufficient condition for *MMD* being small enough: for any $\varepsilon > 0$, we have $MMD \leq \varepsilon$ whenever $\|\mu_e - \mu^{\widetilde{\pi}}\|_2 \leq \varepsilon$.

3.2.4 The Algorithm

Using this property, we give the description of Apprenticeship Learning [1]. One way to view this algorithm is as follows. The algorithm iteratively generates policies until the difference in feature expectations between any of the generated

policies and an expert policy becomes small enough. The policy that matches well the expert policy in feature expectations is guaranteed to have *MMD* small enough under some worst-case R_w (that can be computed in closed form when the matched policy is specified). Another perspective is that the algorithm generates a reward function that places a high expected reward to the expert and low to the generated policies. The loop continues until there is no reward function that can make the expert perform far better than all of the generated policies. This implies there exists at least one policy among Π whose performance is close enough to the expert policy *in worst case*. According to the algorithm, the best policy must be selected among Π. One suggested method is to produce a stochastic policy by taking a convex combination of feature expectations of the generated policies. Note there is no guarantee that the reward function is the same as the reward function from which the expert policy had originally been derived.

3.2.5 Equivalent Quadratic Programming Formulation

Also line 5 can be formulated into a quadratic programming problem (quadratic due to the l2 norm constraint):

$$\max_{t,w} t \tag{3.9}$$

$$\text{s.t. } w^\top(\mu_e - \mu^{\pi^{(j)}}) \geq t,\ j \in [i-1] \tag{3.10}$$

$$\|w\|_2 \leq 1 \tag{3.11}$$

Finally, notice *line6* is usually the computational bottleneck of apprenticeship learning. It contains a full-fledged reinforcement learning problem requiring to solve a new MDP at each iteration.

3.3 What Makes the Batch Constraint Challenging

In this section, we explain the reason AL does not work out of the box under the batch constraint and discuss what needs to be adapted. How we will adapt it will be left to the next chapter. In the previous section, we showed AL monitors its policy improvement progress through feature expectations of the set of policies it generates. Under the batch constraint, we cannot directly evaluate $\mu^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T}\gamma^{t}\varphi(s_t,a_t) \mid S_0 = s, A_0 = a\right]$, for we cannot collect additional Monte Carlo samples according to π. Thus, we must somehow estimate $\mu^{\pi}(s,a)$ with the samples generated by a different policy. Furthermore, it is true there exist other family of IRL methods that do not explicitly use feature expectations. However, such methods still assume the existence of a simulator or the knowledge of environment dynamics. Thus, they do not work out of the box in the batch setting.

The general challenge calls for an appropriate name. We believe it is an off-policy evaluation (OPE) [56], the problem of evaluating a policy (evaluation) using the samples collected by another policy (behavior). Below, we limit ourselves to the off-policy estimation of feature expectations but the discussion holds valid generally for other IRL methods that require OPE.

Amid the challenge, we briefly explore the problem of estimating feature expectations of an evaluation policy, π_{new}, using a batch of samples generated by another policy π_{old}. In the IRL context, π_{new} represents a policy generated at each iteration and π_{old} an expert policy. In essence, feature expectations tell us how similar two policies are where the similarity is measured by the feature space expected to be encountered by each policy under some fixed $\mathcal{M}$. Feature expectations are closely related to occupancy measures that will be discussed later in the Kernel-based Imitation Learning chapter. Below, we explore a few possible ways to estimate feature expectations and potentially the approaches can be mixed together.

Model-free Approach: We cast the estimation problem as an off-policy estimation problem, similar to the estimation of an action-value function $Q^\pi(s,a)$ [27]. The difference is that $Q^\pi(s,a) \in \mathbb{R}$ whereas $\mu^\pi(s,a) \in \mathbb{R}^d$. We can interpret i-th entry of $\mu^\pi(s,a)_{(i)}$ to be associated with an action-value function—each different. This view allows us to consider a wide range of model-free value function estimation algorithms. [27] used the formulation of the LSTD-Q method [30].

Model-based Approach: We tackle the estimation problem by estimating an environment dynamics (the model) T, T_0 and subsequently evaluate feature expectations using the model. It is a nice approach to consider wherever possible since it can potentially reduce variance of the estimation, potentially at the expense of bias [17]. The approach can naturally accommodate Bayesian methods where some prior knowledge of the model is available [48].

Importance-Sampling-based Approach: We borrow a popular method to off-policy evaluation problems: importance-sampling. [56]. Briefly, we can use the relationship $\mathbb{E}_{\pi_{new}}\left[\mu\right] = \mathbb{E}_{\pi_{old}}\left[\frac{\pi_{new}}{\pi_{old}} \cdot \mu\right]$. This assumes we have π_{old} available. In the IRL context, the assumption is not valid so we can take an empirical estimation $\widetilde{\pi_e}$ from D_e. However, importance-sampling-based approaches come with some fundamental, practical challenges of their own, so it is pivotal to consider a way to address them [7].

3.4 Survey of Batch IRL methods

In this section, we briefly review a set of methods that can address the batch constraint, both explicitly by design (the first four) or implicitly (the rest). As explained in the previous section 3.3, the batch constraint is closely related to an off-policy evaluation problem. Naturally, one may find potentially useful tactics in the broad field of off-policy learning [13, 55, 58], a class of algorithms that aim to produce monotonically improved policies given a fixed set of data at each

iteration. The emphasis is on policy and hence, such tactics tend to be more applicable in the imitation learning context. Relatively speaking, there are few IRL methods applicable under the batch constraint. We believe the main reason is the intrinsic difficulty of the IRL problem plus the additional challenge posed by the batch constraint. To make the batch IRL problem more tractable, strong assumptions have often been made such as knowledge of environment dynamics or an additional dataset generated by non-experts.

3.4.1 Least Squares Temporal Difference (LSTD-μ)

LSTD-μ [27] addresses the batch constraint as an off-policy value function estimation problem. While leaving the method theoretically compatiable with any off-policy, model-free value function estimation methods, the author suggested Least Squares Temporal Difference (LSTD) [29]. Recall the conditional feature expectations of a policy is $\mu^\pi(s,a) = \mathbb{E}_\pi\left[\Sigma_{t=0}^{T}\gamma^t\varphi(s_0,a_0) \mid S_0 = s, A_0 = a\right] \in \mathbb{R}^d$. Let $\mu_i^\pi(s,a)$ be the i-th component.

The key observation is that the formulation is precisely the action value function of a policy under some fixed $\mathcal{M}$ whose reward function is defined to be φ_i where φ_i is the i-th component of a predefined feature map: $Q_i^\pi(s,a) = \theta_i^\top \mu_i^\pi(s,a)$ where θ is the parameters of a feature expectation estimator. Utilizing this view, the method searches for a value function that estimates feature expectations and in turn, finds a reward function that can explain expert behaviors.

A key assumption is that the pre-chosen feature map φ is rich enough in the sense that the hypotheses space of value functions and the reward hypotheses mostly overlap. As will be discussed in more detail in the experiment section of the next chapter, LSTD-μ inherits the fundamental problem of *LSTD*-based methods: the sensitivity to input data distribution. According to our own experiments and the original author's experiments [1], LSTD-μ in practice requires

[1] https://github.com/edouardklein/RL-and-IRL

a non-expert dataset to obtain φ and save the method from being an under-determined linear system.

3.4.2 Structured Classification (SCIRL)

Structured Classification-based IRL (SCIRL) [28] was proposed by the same author with the aim to address the computational complexity of the IRL problem. The key idea is to cast the IRL problem as a multi-class classification problem (supervised learning). Concretely, consider a linearly parameterized score function $f_w(s,a) \triangleq w^T \mu^{\pi_e}(s,a) \in \mathbb{R}, s \in \mathcal{S}, a \in \mathcal{A}$. Notice the feature expectations are conditioned on the expert policy.

To predict the expert action (or equivalently to classify), a decision rule is derived from solving $\pi(s) \in \arg\max_{a \in \mathcal{A}} f_w(s,a)$ for a fixed s. There is no restriction on the choice of a training algorithm for the classifier. For instance, a multi-class Support Vector Machine can be used [16]. After f_w is trained, we compose a reward function $r_w(s,a) = w^\top \varphi(s,a)$. We refer the reader to the analysis section in [28] to read the justification of the reward composition. The big idea is that π_e can be made optimal up to the precision we chose under r_w.

To estimate $\mu^{\pi_e}(s,a)$ where $a = \pi_e(s)$, it suggests LSTD-μ and for when $a \neq \pi_e(s)$, it suggests a heuristic where $\mu^{\pi_e}(s, a \neq \pi_e(s)) = \gamma \mu^{\pi_e}(s, \pi_e(s)), \gamma \in [0,1)$. The heuristic dictates that the feature expectation of an expert policy for a *non-expert* action differs from that of an expert action up to a multiplicative factor of γ. As will be discussed in the experiment section of the next chapter, the heuristic approach does not work generally. One problem is that the multi-class classification view does not take into account the environment dynamics that can lead to a problem [47]. Also the theoretical analysis assumes some sufficient richness of φ but there is no specific way suggested for finding a good φ that can lead to another issue [33].

3.4.3 Boosted and Reward-regularized Classification (RCAL)

RCAL [42] took a similar approach as SCIRL, addressing the IRL problem via a multi-class classification problem. The key difference is the notion of a large-margin error. The classification loss function is taken to be $L(w) = \mathbb{E}_{\mathcal{S}\times\mathcal{A}}\left[\max_{a'\in\mathcal{A}}[f_w(s,a') + l(s,a,a')] - f_w(s,a)\right], f_w = \arg\min L(w)$ where the expectation is taken to be the visitation distribution induced by the decision rule of f_w. The motivation is to make the method more robust by finding a score function f_w that assigns to expert actions a score higher at least by some margin $l(s,a,a')$ than the rest of the actions. In order to consider the environment dynamics, the authors suggested the existence of an additional dataset that contain transition samples. We believe the assumption, in practice, is strong. To bypass the need for choosing a feature map, the authors suggested combining their method with boosting [15]. To retrieve a reward function, they use $R(s,a) = f_w(s,a) - \gamma \max_{a\in\mathcal{A}} f_w(s,a)$ that follows from the Bellman Optimality Equation assuming f_w is optimal. The large-margin approach has proven to be useful also in a non-batch setting [18].

3.4.4 Estimation of Rewards and Dynamics (SERD)

Simultaneous Estimation of Rewards and Dynamics (SERD) is an extension of the Maximum Causal Entropy IRL Algorithm [4]. The difference is to integrate the learning of environment dynamics explicitly under the batch constraint. They set $\mathcal{R}, T_a, T$ as inference targets where T_a is the dynamics believed by the agent and T the true environment dynamics. By doing so, they aim to increase the sample efficiency and the accuracy of the reward function estimation. They suggested a gradient-based maximum likelihood approach where all parameters $\Theta = (\theta_R, \theta_{T_a}, \theta_T)$ are simultaneously updated.

Unlike the classification-based approach, it still needs to solve a reinforcement learning problem repeatedly. An advantage is that the learned model may be useful in a transfer learning setting. As will be discussed in more detail in the discussion section of the next chapter, we think learning the dynamics can serve

as a powerful force but one must be careful in deciding to what degree one will use the dynamics model. For instance, we noticed actively using a learned dynamics model for feature expectation estimation can destabilize the training and decrease the overall performance.

3.4.5 Other Methods

A family of methods that we do not cover in this thesis but still matter depending on the context is imitation learning. If one aims to learn an optimal policy—and it is the only objective—imitation learning is generally computationally efficient and easier to implement [20]. However, if one aims to harbour other objectives such as the analysis of expert motivation or the transfer setting [11, 41].

Another family of IRL methods to consider, though only potentially applicable for the batch constraint, is the relative entropy [5] or maximum likelihood [2] based methods. The key formulation is the connection between a distribution of trajectories (derived from a policy) and a reward function (that induces the policy): $\mathbb{P}(\tau \mid w) \propto T_0(s_0) \exp\left(\sum_{(s,a)\in\tau} w^\top \varphi(s,a)\right) \prod_{t=1}^{T}(s_{t+1} \mid s_t, a_t)$. By formulating the problem, one can enjoy a principled way of dealing with stochastic expert policies [59]. The approaches are usually formulated under the feature expectation matching condition $|\mu^\pi - \mu^{\pi_e}| < \varepsilon$, but it seems this is not absolutely necessary depending [2].

Alone we can do so little; together we can do so much.

Helen Keller

4

Deep Successor Feature Network

4.1 Introduction

This chapter introduces a novel method that naturally addresses key challenges of the batch inverse reinforcement learning setting. The method is a two-step hybrid method in the sense that it combines imitation learning and inverse reinforcement learning. The imitation learning step provides an initial policy and a feature map, where the two components naturally help warm-start the inverse reinforcement learning step. The two components are typically assumed to be given a priori in IRL literature and in practice, requires non-trivial amount of domain knowledge to manually engineer them [33].

Regarding the IL part, this chapter shows that the acquisition of the two components via imitation learning strongly supports inverse reinforcement learning in increasing convergence rates and improving reward hypotheses

spaces. This chapter proposes Transition Regularized Imitation Learning (TRIL), a regularization scheme for a neural-network-based imitation learning model.

Regarding the IRL part, this chapter introduces Deep Successor Feature Network (DSFN), a neural-network-based model that estimates feature expectations in the batch setting, the most crucial contribution of the entire thesis. The chapter discusses connections to deep successor features. In the experiments, we compare the performance of TRIL and DSFN to baseline batch IRL methods.

4.2 Preliminaries

4.2.1 Problem Setting

We will use the same notations as we defined and have used so far. We consider a stationary, infinite-horizon MDP $\mathcal{M}$ equipped with $\mathcal{S}, \mathcal{A}, \mathcal{R}, T, T_0, \gamma$. We do not make assumptions about the continuity of state and action spaces, for the method we propose in this work is theoretically free to choose either. The experiments we share in this work, however, deal with continuous state space and discrete action space. We assume a stationary, stochastic expert policy π_e that generates a set of demonstrated trajectories $D_e = \{\tau \mid \tau \sim \pi_e\}, |D_e| = N \in \mathbb{N}$ under $\mathcal{M}$. Similar to [1], we assume the expert acts with respect to some unknown normalized *linear* reward function, $\mathcal{R}_e \in \mathcal{R}_{linear}$ composed with unknown parameters $w_e \in \mathbb{R}^d, \|w_e\|_2 \leq 1$ and with unknown feature map $\varphi_e : \mathbb{S} \times \mathbb{A} \mapsto [0,1]^d$. Let μ^π denote a feature expectation of π, $\mu^\pi \stackrel{\Delta}{=} \mathbb{E}_\pi\left[\Sigma_{t=0}^{T} \gamma^t \varphi(s_0, a_0)\right]$ and $\mu^\pi(s,a)$ a conditional feature expectation $\mu^\pi(s,a) \stackrel{\Delta}{=} \mathbb{E}_\pi\left[\Sigma_{t=0}^{T} \gamma^t \varphi(s_0, a_0) \mid S_0 = s, A_0 = a\right]$, both under fixed $\mathcal{M}$.

4.2.2 Batch Constraint

In addition to the problem formulation above, we add the batch setting constraint. We assume an agent is given D_e, that it does not know environment dynamics T, T_0 and that it cannot interact with the environment $\mathcal{M}$. The

no-dynamics and no-interaction constraints mean we lose the ability to evaluate any policy directly. Instead, we must evaluate one policy using samples generated by a different policy D_e. Hence, the batch constraint implicitly requires an off-policy policy evaluation [56]. As we reviewed in the Batch IRL chapter that feature expectations are a crucial metric in many of the foundational IRL methods. Lastly, the goal is to find a reward function that can reproduce a behavior close enough to the expert demonstrations. Simultaneously, we find a policy whose performance is close enough to π_e under *any* linear reward function.

4.3 Deep Successor Feature Network

In this section, we introduce Deep Successor Feature Network (DSFN). In estimating feature expectations, we adopt the model-free formulation [27] and parameterize the feature expectation estimator using a deep neural network. Moreover, we adopt Apprenticeship Learning (AL) [1] as an example of feature-expectation-based IRL methods. As DSFN does not require assumptions specific to AL, DSFN is generally compatible with any IRL methods that use feature expectations.

4.3.1 The Algorithm

Define s_T as the terminal state. Let $\mu_\theta^\pi(s,a)$ denote the feature expectation estimator (DSFN) parameterized by a neural network (θ) for an evaluation policy π. We aim to learn $\mu_\theta^\pi(s,a) \approx \mu^\pi(s,a), \forall(s,a)$. We train the estimator using a stochastic gradient descent algorithm on the loss derived from the Bellman equation 4.2. Fix π, φ. We set the Bellman targets (the ground-truth targets to regress on) $\forall(s,a,s') \in D_e$ in Equation 4.1

$$y^\pi_{(s,a,s')} = \begin{cases} \varphi(s,a) & \text{if } s' = s_T \\ \varphi(s,a) + \gamma \mathbb{E}_{a'}[\mu_\theta^\pi(s',a')] & \text{otherwise} \end{cases} \tag{4.1}$$

Algorithm 3 Batch Apprenticeship Learning

Input: π_0 (Initial Policy), φ (Feature Map)
Parameter: $w \in \mathbb{R}^{d_w}$, θ
Output: $w_{(n)}$

1: **for** i = 0 : n **do**
2: Evaluate $\mu_\theta^{\pi_{(i)}}$ using DSFN (Algorithm 2)
3: Compute a reward function $w_{(i)}$ by solving max-margin QP

$$w_{(i)} = \min_{w \in \mathbb{R}^{d_w}} ||w||^2$$

$$\text{s.t.} \quad w^T \mu_j^\pi \leq w^T \mu_e^\pi + 1, \; \forall j \in \{1, 2, \dots (i-1)\}$$

4: Optimize MDP (any solver) with respect to $r_{w_{(i)}}(s,a) = w_{(i)}^T \varphi(s,a)$ to obtain $\pi_{(i+1)}$.
5: **end for**
6: **return** $w_{(n)}$

Notice $y^\pi_{(s,a,s')}$ is a function of π and varies wits respect to the policy. We use Mean Squared Error (MSE) loss. And the loss and its gradient $\forall (s,a,s') \in D_e$ can be computed:

$$\begin{aligned} \mathcal{L}(\theta, \pi) &= \frac{1}{2} \mathbb{E}_{(s,a,s')} \left[\; ||\mu_\theta^\pi(s,a) - y^\pi_{(s,a,s')}||^2 \; \right] \\ &\approx \frac{1}{N} \sum_{i=1}^{N} ||\mu_\theta^\pi(s_i, a_i) - y^\pi_{(s_i,a_i,s_i')}||^2 \qquad (4.2) \\ \nabla_\theta \mathcal{L}(\theta, \pi) &= \mathbb{E}_{(s,a,s')} \left[\; (\mu_\theta^\pi(s,a) - y^\pi_{(s,a,s')}) \cdot \nabla \mu_\theta^\pi(s,a) \; \right] \end{aligned}$$

The expectation is taken with respect to some visitation distribution induced by π, T, T_0. Notice the training procedure of DSFN is analogous to that of Deep-Q networks [36] with the subtle difference that DSFN performs policy evaluation while DQN policy improvement. Also we assume we had made available a validation dataset from D_e, since we cannot collect additional data to monitor the out-of-sample loss under the batch setting. We terminate our algorithm when validation loss $\mathcal{L}_{\text{val}}$ drops below some pre-defined threshold of $\delta > 0$ (Algorithm

Algorithm 4 Deep Successor Feature Network (DSFN)

Input: D_e (Data), φ (Feature Transformer), γ, π, δ
Parameter: θ
Output: μ_θ^π

1: Feed D_e to Experience Replay Buffer (ERB).
2: Initialize θ for $\mu_\theta(\pi, \gamma)$
3: **while** $\mathcal{L}_{\text{val}} > \delta$ **do**
4: Sample a batch $B = \{(s, a, s')\}$ from ERB.
5: Set $\{y^\pi_{s,a,s'}\}$ for the batch given φ, γ as in Eqn. (4.1).
6: Compute a mini-batch gradient of B.
7: Update θ with gradient descent using Eqn. (4.2).
8: **end while**
9: **return** θ

2).

4.4 Transition-Regularized Imitation Learning

In this section, we introduce another neural network model that warm-starts IRL. By warm-starting, we mean the process of preparing ingredients useful for performing IRL. The ingredients are an initial policy π_0 and a feature map φ. We briefly motivate the necessity.

4.4.1 Necessity of Warm-starting

A careful reader would notice, in Eqn. (4.2), that the expectation is approximated with respect to D_e as opposed to $D' \sim \pi$. This can lead to a high bias when π is significantly different from π_e, for the training data support could be nearly disjoint (i.e. $D \sim \pi, D \cap D_e \approx \emptyset$). Since one cannot collect additional transitions in batch settings, our gradient updates for μ^π can be heavily biased. Consequently, IRL with DSFN may fail to converge. Thus, an initial policy must be similar to π_e already so that μ^{π_0} can be accurately evaluated on the part of state-action space

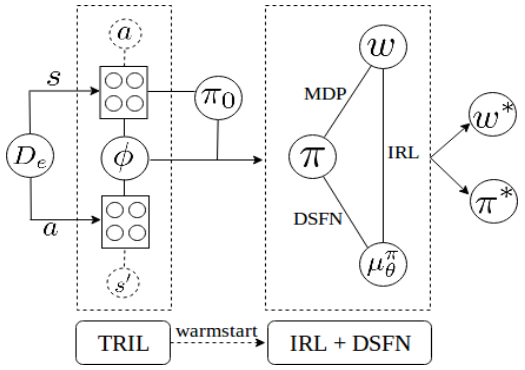

Figure 4.4.1: Overview of TRIL+DSFN: TRIL is a dual-task neural network that jointly predicts expert action and next states. TRIL warm-starts IRL by providing an initial policy π_0 and a feature map φ (the shared hidden layers). Using this feature space in IRL, DSFN provides off-policy estimations of feature expectations of any candidate policy, which is essential to update the reward function in max-margin IRL. We used Apprenticeship Learning [1] to optimize the reward function (IRL) and DQN [36] to obtain an optimal policy (MDP).

seen in D_e. For instance, a uniform random policy would not be suitable. Furthermore, IRL is highly sensitive to the choice of a feature map and a good feature map is hard to manually engineer [33]. Hence, it would be useful if one had a procedure of engineering a feature map without manual efforts.

4.4.2 The Algorithm

We introduce Transition-Regularized Imitation Learning (TRIL), a regularized supervised learning classifier. The model is designed to obtain a near-expert initial policy while simultaneously deriving a good feature map. TRIL is a dual-task neural network, jointly trained to predict the expert's action given state and the system's next state transition given state and expert action. The idea of regularizing with next state predictions is similar to [39]. The idea of using a neural network as a feature map provider is similar to [53].

The training procedure is similar to a multi-task supervised learning classifier [9]. Let θ_{π_0} be the parameters of TRIL and L_{ce} be the cross entropy loss in

predicting expert's action and L_{mse} be the MSE loss on next state predictions conditioned on current state and the expert's action. The training samples are considered to be independent transitions $(s, a, s') \in D_e$. Let $\lambda \in \mathbb{R}$ be the regularization coefficient that controls the strength of the regularization. The network is trained using the following loss: $\forall (s, a, s') \in D_e$

$$L(\theta_{\pi_o}) = L_{\text{ce}}(a, \pi_o(s)) \; + \lambda L_{\text{mse}}(T_{\pi_o}(s, a), s') \tag{4.3}$$

Algorithm 5 Transition Regularized Imitation Learning (TRIL)

Input: D_e (Expert Demonstrations)
Parameter: θ
Output: π_o (Initial Policy), φ (Feature Map)

1: Feed D_e to Experience Replay Buffer (ERB).
2: Initialize a dual-channel neural network f_θ.
3: **while** $\mathcal{L}_{\text{val}} > \delta$ **do**
4: Sample a batch $B = \{(s, a, s')\}$ from ERB.
5: Compute a mini-batch gradient of B according to 4.3.
6: Update f_θ and using Adam Optimizer [26]
7: **end while**
8: Set the shared hidden layers of f_θ to be φ
9: Set the action-prediction channel of f_θ to be π_o
10: **return** π_o, φ

4.5 Experiments

In this section, we report the empirical evaluations of our model (TRIL+DSFN). To the end, we designed two sets of experiments to study following questions:

- How well does our model predict in expert actions, compared to other baseline methods?
- Is TRIL really necessary? When does it make sense to use TRIL?

- What empirical issues do we notice when applying TRIL+DSFN?

The first set of experiments includes three benchmark control environments: Mountaincar-v0, Cartpole-v0 and Acrobot-v1 [1]. The second set of experiments includes the prediction of optimal treatment for Sepsis patients in Intensive Care Units (ICU). To measure a validation accuracy (in the control experiments) or to estimate out-of-sample accuracy (in the ICU experiment), we split the training datasets into a 70 to 30 ratio. Since there exist simulators for the control environments, we measure the performance of the final policies obtained from our model by rolling them out in the simulators. That is, we relax the batch constraint only at the test time. On the contrary, there is no generally acceptable simulator for modelling the behaviors of Sepsis patients in the ICU [44]. Hence, we use a carved-out test data to evaluate the performance. We adopted Apprenticeship Learning as our IRL framework.

In our experiments, we employed a total of three neural networks: TRIL, DSFN and Double DQN (the MDP solver in the IRL phase). We reported hyper-parameters of the three deep models at the end of this section **??**. To our knowledge, there is no peculiar element in the architecture of the neural networks, except for one. Following [6], TRIL and DSFN included a Gaussian layer, as the final hidden layer. The Gaussian layer learns both the mean and standard deviation of the quantity it is designed to predict such as expert actions, next states or feature expectations.

4.5.1 Baselines

As for the baseline methods, we included random policy, LSTD-μ and SCIRL—we described the latter two in the previous section. Both LSTD-μ and SCIRL require a feature map by assumption. Unlike DSFN that inherited a feature map from TRIL, the two methods used feature maps obtained separately from the state samples uniformly sampled from the environment. The two methods could have also benefited from TRIL but we decided not to warm-start

[1] https://github.com/openai/gym

with TRIL to keep their performances in line with their original results. However, for the real-world ICU experiment, we warm-started all baseline methods (except for random policy) with TRIL. For instance. in MountainCar-v0, we used a Gaussian kernel of 25 components for $\varphi(s)$. Subsequently we onehot-encoded $\varphi(s)$ based on the 3 actions to represent $\varphi(s, a)$ so its dimension becomes 75. For Acrobot-v1 and Cartpole-v0, we used RBF Kernel of 100 components (25 components each $\gamma = 0.1, 0.5, 1.0, 5.0$).

environment	dim(s)	dim(a)
MountainCar-v0	2	3
Cartpole-v0	4	2
Acrobot-v1	6	3
Sepsis	46	5

Table 4.5.1: Benchmark Environments: the state and action space dimensions on OpenAI gym and Sepsis benchmarks

4.5.2 Classical Control Tasks

We first obtained the optimal policies (experts) via Double DQN [57]. Technically speaking, the policies are near-optimal in the sense that the performance of the policies exceed the pre-defined performance levels of the three tasks (OpenAI provides them). We used the policies to sample demonstration trajectories of varying number of episodes (1, 10, 100, and 1000) that in turn constitute the training batch datasets D_e. After this point, we adopted the batch constraint: we removed the access to the simulator. All experiments must be done in batch, model-free setting.

We set the maximum of 10 iterations with two stopping conditions. The first condition was at the feature expectation margin of 0.1 (that makes sense we assume φ, w are normalized, bounded). The second condition was when the validation accuracy of action prediction for the two consecutive iterations goes lower than 5%. We found the latter stopping condition to be useful in keeping the

training loop stable. Unlike typical inverse reinforcement learning routines, there is no correcting mechanism that's based on the ground-truth information (typically achieved by on-policy evaluation via simulators or the knowledge of environment dynamics). Thus, we needed ways to estimate the training progress still under the batch setting; otherwise, we noticed the training may go unstable.

The Experimental Results

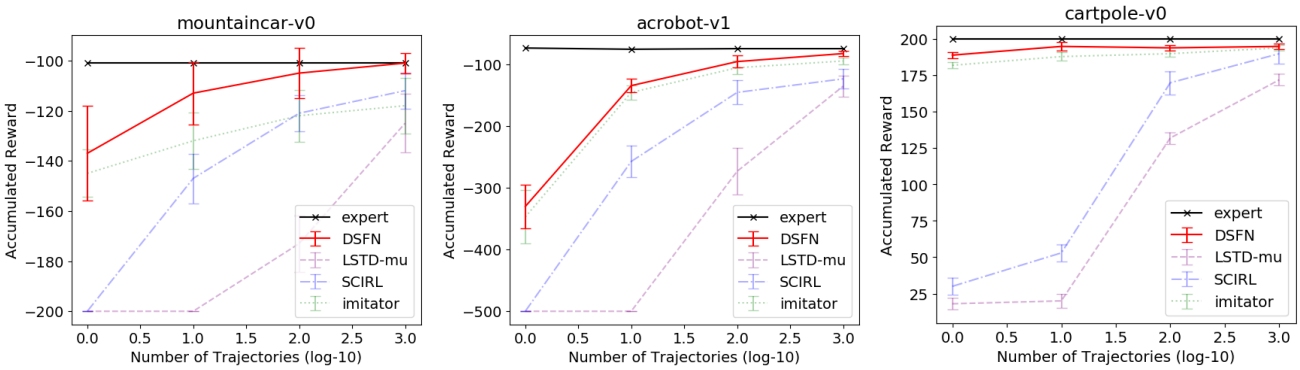

Figure 4.5.1: Classical Control Experiment of TRIL+DSFN: TRIL+DSFN closely matched the original performance of the experts. It also showed a greater sample efficiency across all episode lengths (note *log* scale on the x-axis).The performance has been averaged over 5 trials where the error bar shows one standard error. More demonstration trajectories lead to less estimation errors.

We reported the experimental results in Figure 4.5.1. We remark on two things. First, TRIL+DSFN outperformed the baselines in sample-efficiency. Across all of the three tasks and all episode lengths, TRIL+DSFN outperformed the baselines in performance—reaching near-expert performance. We observed that LSTD-μ performed poorly because of its high sensitivity to the coverage and distribution of the input data, a well-known issue in LSTD-based methods. In poor coverage and distribution of the input data, LSTD-methods tend to lead to an under-determined system. We observed SCIRL training to be not stable. We believe this is due to its reliance on a LSTD method for feature expectation

estimation. Moreover, we believe it may be attributable to the wrong application of the heuristic terms in SCIRL. Second, we observed TRIL+DSFN successfully found reward functions under which its policies and the expert policies are matched closely in performance. Notice the TRIL's performance alone (Imitator) is already quite good, making one wonder "is DSFN necessary?". We shall remark on this issue in more detail in the Discussion section. As a matter of fact, the vanilla supervised learning imitator, that is not transition-regularized, performed much more poorly than TRIL.

4.5.3 Sepsis Treatment Tasks

In this section, we report the performance of TRIL+DSFN on a real-world batch data.

About Sepsis

Sepsis is a leading cause of cost and mortality in Intensive Care Units (ICU), killing 258,000 Americans every year [34]. [44] showed Deep Reinforcement Learning can be used to predict optimal IV-fluid and vasopressor intervention treatment for Sepsis patients. In this section, we not only focus on obtaining a policy that closely matches the demonstrated expert behavior (the clinician's historical interventions) but also analyze the reward functions obtained in IRL. Through the reward function analysis, we reported that the IRL-learned reward functions are on par with standard clinical intuitions. Concretely, we analyze reward functions with respect to vasopressor administration. Vasopressor is deemed a critical clinical intervention to react to the onset of Sepsis [34]. Furthermore, we note that this type of reward analysis effort can be useful since it may help us investigate the underlying motivation of experts: what are clinicians optimizing for with these vasopressor interventions?. It is often hard to expect human experts to answer the question directly.

1. Index Measures: Shock Index, Elixhauser, SIRS, Gender, Re-admission, GCS - Glasgow Coma Scale, Age

2. Lab Values: Albumin, Arterial pH, Calcium, Glucose, Hemoglobin, Magnesium, PTT - Partial Thromboplastin Time, Potassium, SGPT - Serum Glutamic-Pyruvic Transaminase, Arterial Blood Gas, BUN - Blood Urea Nitrogen, Chloride, Bicarbonate, INR - International Normalized Ratio, Sodium, Arterial Lactate, CO_2, Creatinine, Ionised Calcium, PT - Prothrombin Time, Platelets Count, SGOT - Serum Glutamic-Oxaloacetic Transaminase, Total bilirubin, White Blood Cell Count

3. Vital Signs: Diastolic Blood Pressure, Systolic Blood Pressure, Mean Blood Pressure, $PaCO_2$, PaO_2, FiO_2, Respiratory Rate, Temperature (Celsius), Weight (kg), Heart Rate, SpO_2

4. Intake and Output Events: Fluid Output - 4 hourly period, Total Fluid Output, Mechanical Ventilation, IV Fluids

The Experimental Setting

The Multiparameter Intelligent Monitoring in Intensive Care (MIMIC-III v1.4) database [23] includes a cohort of 17,898 patients satisfying Sepsis-3 criteria. We followed the experimental setup of [44]. We considered a total of 46 observational features (patient vitals and lab measurements) including other important interventions such as mechanical ventilation and IV fluids at each time-step. Hence, the state space was $\mathcal{S} \in \mathbb{R}^{46}$. Furthermore, we had the discrete action space $|\mathcal{A}| = 5$, where each action amounted to choosing one of 5 vasopressor dosage bins. The bins were chosen based on the empirical distribution of vasopressor administered (quartiles). We consider in-hospital mortality and leaving the ICU (alive) as the absorbing states. Each patient's treatment trajectory includes an expert demonstrated trajectory (an episode). The observed trajectory lengths were at most 20 steps (about 80 hours of ICU stay since the data was collated over 4 hour time windows). We used a discount factor γ of 0.99.

For a fair comparison, we provided warm-started all of the algorithms except the random policy with TRIL—the same initial policy and the same feature map.

Also, due to the batch constraint, the reported performance was not measured on a simulator or on a real-world verification. As such, the performance metric was defined to be simply expert action prediction. This is a somewhat misleading metric as it fails to account for the compounding effect of small action prediction errors over trajectories [47].

Method	Top-1 matching	Top-3 Matching
TRIL+DSFN	$79 \pm 5\%$	$90 \pm 3\%$
TRIL+LSTD-μ	$39 \pm 4\%$	$69 \pm 3\%$
TRIL+SCIRL	$36 \pm 5\%$	$61 \pm 4\%$
Random	$20 \pm 1\%$	$49 \pm 6\%$
TRIL (regularized)	$80 \pm 2\%$	$91 \pm 1\%$
IL (not regularized)	$29 \pm 5\%$	$58 \pm 4\%$

Table 4.5.2: Expert Action Matching Accuracy We measured the clinician (expert) action classification accuracy on the test dataset. The performance was measured on the test dataset over three trials. Top-1 matching checks whether policy's best action matches clinician actions and the Top-3 matching whether the clinician actions are included in the top 3 choices of the policy.

The Experimental Results

As can be seen in Table 4.5.2, our DSFN showed approximately 80% action-matching. We observed DSFN outperformed the baseline methods. The performances of LSTD-μ and SCIRL lagged behind, as we believe due to the sensitivity to the coverage and the input data distribution. We found adding a Gaussian layer, therefore representing a stochastic policy, to be crucial. We hypothesize this is owing to the intrinsic difficulty of managing Sepsis: there is not yet a general agreement on what constitutes the optimal treatment for Sepsis patients among clinicians. Our neural networks account for such uncertainty by trying to learn standard deviations as well. Also, we would like to remark that some action prediction errors may have been attributable to the discretization of the action space.

We noticed a significant advantage of having TRIL in place. As can be seen in Table 4.5.2 for sepsis, TRIL outperformed the unregularized baseline. In our sepsis experiment, obtaining an initial policy from TRIL was necessary for DSFN to perform well. DSFN without TRIL did not converge—so were LSTD-μ and SCIRL. We think for a task as complex and uncertainty-abound as sepsis management, it is essential to warmstart DSFN with TRIL. Notice the unregularized version of imitation learning did not perform as well, suggesting that not all imitation learning networks will be helpful for warmstarting.

We observed the reward function learned by TRIL+DSFN agrees with clinical practice. Figure 4.5.2 shows the relationship between rewards and patient vitals based on patient conditions. The dashed line indicates the learned reward conditioned on no vasopressor; the solid line with a high dose of vasopressor. It is known that Sepsis patients are correlated with hypotension, high heart rate and low platelet count [46]. We observed that the learned reward function reflect on this by giving low rewards for taking no action for patients of low BP, high heart rate, or low platelet count. And the opposite for patients with Septic shock.

4.6 Frequently Asked Questions

In this section, we aim to address common questions the readers may have.

Is it worth estimating a reward function when TRIL already performs so well? We argue that obtaining a reward function via DSFN is still valid even when TRIL's performance is near optimal. As discussed in [11, 41], (linear) reward functions are not only easier to interpret but also (potentially) more generalizable. Hence, it depends on the requirements of a task. If the goal is simply to retrieve a good policy, TRIL or other imitation learning methods would suffice.

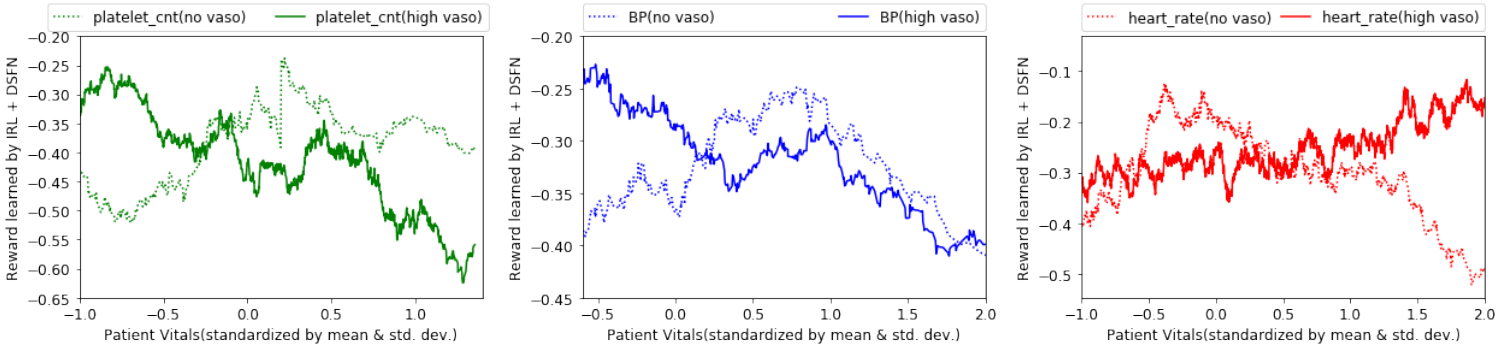

Figure 4.5.2: Analysis of a DSFN-TRIL-derived Reward Function. We plotted a few patient vitals (standardized) against the rewards assigned by the learned reward function. On the left, we observed high rewards on the high-vasopressor-region for low platelet patients and low vasopressor for low platelet counts. In the middle, we observed higher rewards assigned to high vasopressor for low BP patients. Similarly, to the right, we observed high rewards assigned to high vasopressor for high heart-rate patients. This is in line with the general knowledge that the onset of Sepsis is correlated with low blood pressure, low platelet counts and high heart rate.

IS IT WORTH WARMSTARTING WITH TRIL? Yes, we tried initializing IRL with DSFN only, various initial policies and various feature maps (such as Gaussian kernels). In most cases, DSFN did not converge with high inaccuracies in estimates.

WHY NOT USE THE LEARNED MODEL? It is true we acquire an environment dynamics model that predicts the next states given $(s, \pi_e(s))$. We tried integrating the learned model in the IRL phase to estimate feature expectations, which is valid theoretically. Unlike the work of [17] that claims the benefit of directly integrating a learned model in IRL, our empirical results suggested otherwise. We believe this is because learning environment dynamics is hard for complex tasks and a mis-specified dynamics model can severely influence the estimation.

WHY IS TRANSITION REGULARIZATION USEFUL? As analyzed in [47], it is not wise to naively reduce imitation learning to a supervised learning problem. The

Hyperparameters	TRIL	DSFN	DQN
number of hidden layers	2	2	2
hidden node size	128	64	128
max training iterations	50000	50000	30000
activation function	tanh	tanh	tanh
optimizer	Adam	Adam	Adam
adam epsilon	1e-4	1e-4	1e-4
adam learning rate	3e-4	3e-4	3e-4
mini-batch size	64	32	64
λ (regularization)	1.4	-	-
state normalizer	Y	Y	N
PER	N	N	Y
PER α	-	-	0.6
PER β_0	-	-	0.9
moving average (DDQN)	-	0.01	0.01
discount rate	0.99	0.99	0.99
stopping condition (validation)	5e-3	5e-3	1e-2

Table 4.5.3: The hyperparameters of the three neural network models

training samples are correlated and hence, even small errors can accumulate to become unmanageable. Regularization with next state predictions seem useful because it encourages an algorithm to prefer the subspace of hypotheses space that agrees with the underlying environment dynamics. Similar arguments have been made in [39, 40, 42, 53].

4.7 Conclusion

Solving the IRL problem under the batch constraint is hard. Let us briefly recite the challenges generally associated with IRL and the challenge peculiar to the batch constraint. The first type of challenges includes computational costs, sensitivity to the choice of feature maps and the intrinsic ambiguities. TRIL effectively addresses the first two challenges by encouraging a faster convergence (via an improved initial quality) and providing a rich feature map, encoded by a

deep neural network. Although we did not show how to address the intrinsic ambiguity in the experiments, it does not mean TRIL+DSFN cannot resolve it. For instance, since TRIL+DSFN is not coupled with Apprenticeship Learning [1], Maximum Entropy IRL [59] can also be combined with it to address the policy-demonstration ambiguity.

The second type of challenges, peculiar to the batch constraint, includes the need for estimating feature expectations. We showed DSFN effectively caters to the need by using a neural network. In general, batch inverse reinforcement learning, or more generally Learning from Demonstrations under the batch constraint is an actively researched area. TRIL+DSFN is one of the first batch IRL methods that have been shown to work well in real-world, large-scale problems. The challenges it addressed are both of theoretical and practical importance.

We observed preparing a good initial policy is crucial for solving the batch IRL problem—especially so when the amounts of expert demonstration are small. When the initial policy is drastically different from the expert policy, the IRL algorithm did not converge due to compounding off-policy evaluation errors.

We observed the importance of regularizing the imitation learning procedure with the task of predicting the next states. We hypothesize the reason why the regularization scheme seems successful is because it encourages an supervised learning optimization algorithm to explore the parameter subspace that agrees with environment dynamics better. One key characterization of a reinforcement learning problem is the correlated training samples and delayed rewards [55].

We observed a learned reward function from TRIL+DSTN agrees with standard Sepsis treatment intuitions agreed among clinicians. One crucial benefit of having a reward function is the possibility of analyzing it to infer the expert motivation. Though it was satisfying to see the learned reward function agreeing with the clinical intuition, we remark that in our model there is not a theoretical guarantee of recovering the reward function that expert is assumed to rely on $\mathcal{R}_e$. In a real-world problem, we even do not know if $\mathcal{R}_e$ lies in $\mathcal{R}_{linear}$ and which feature space φ it is comprised of. It will be a valuable future work to consider

ways to recover the expert reward function by imposing another set of assumptions.

There's always another way.

Courtney Turk

5

Evaluation of Kernel-based Imitation Learning

5.1 Introduction

In this section, we briefly review a recently proposed kernel-based imitation learning approach *without the batch constraint*. Our main interest is to share an empirical evaluation of the method with respect to some of the questions that were not addressed in the original work. Recently in 2018, proposed was Generative Moment Matching Imitation Learning (GMMIL) [25], one of the earliest kernel-based imitation learning methods to our knowledge. It is not well studied yet whether its state-of-the-art results can be easily reproduced and to what extent the method is advantageous. To the end, we briefly review the new method and evaluate it on a synthetic task and OpenAI control benchmarks. We

present the empirical results in terms of the method's reproducibility, stability, sample-efficiency, sensitivity to data dimensionality, and robustness for noise.

GMMIL aims to sit between IRL and IL. First, the method aims to make IRL computationally tractable by not performing the loop in the full scale. Second, its use of kernels may equip the cost function hypothesis spaces with a richer set of choices on the implicit feature maps by the reproducing property. Third, the possible use of *characteric* kernels gives a theoretical guarantee in terms of moment matching of a distribution, making the method better than the simple supervised IL algorithm. Finally, GMMIL maintains a simple formulation and is easier to train than other sophisticated algorithms like the one based on Generative Adversarial Networks [19].

5.2 Preliminaries

In this section, we establish the basic notations. We first remark that we follow the basic notations we have used so far in the thesis, such as $\mathcal{M}$ to denote an MDP, with one difference: instead of considering reward functions $\mathcal{R} \in \mathbb{R}_{\geq 0}$ we consider cost functions $\mathcal{C} \in \mathbb{R}_{\leq 0}$ to respect the notation of the original paper [25]. Let $\mathbb{E}_\pi[c(s,a)]$ be a shorthand for $\mathbb{E}_{s_0,a_0,s_1,a_2,\dots}[\sum_{t=0}^{\infty} \gamma^t c(s_t,a_t)|\pi]$ where $s_0 \sim d(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \in T(s'|s,a)$. It denotes the expected total discounted cost over a trajectory distributed by π. Let π_E always mean the expert (optimal) policy and π_θ the candidate policy we are to optimize. let $\varphi : S \times A \to \mathbb{R}^d$ denote a feature map. Unless otherwise specified, we consider linear cost functions $c \in \mathcal{C} \triangleq \{c \mid c = w^\top \varphi(s,a), w \in \mathbb{R}^d\}$.

5.2.1 Occupancy Measure and Feature Expectations

In this section, we establish the connection between Feature Expectations and Occupancy Measure Matching (refer to [20] for details). IRL aims to learn a cost function that consequently will induce the optimal policy whereas IL focuses on mimicking the expert policy [47], both from expert demonstrations denoted by $D_E \triangleq \{\tau \mid \tau = (s_0, a_0, s_1, \dots)\}$.

We also revisit the notion of occupancy measures we defined in Section 2.2.4. Define the occupancy measure $\rho : S \times A \to \mathbb{R}$ where $\rho_\pi(s,a) \triangleq \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_t = s, a_t = a)] = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s|\pi)$ where $\mathbb{I}$ is an indicator function. This can be interpreted as the unnormalized distribution function over (s,a) given π or as an expected number of visits to (s,a). In the subsequent discussion, we assume ρ to be a finite measure so it is treated as a probability measure on S and A (the detailed justification is in [25]). It allows us to write $\mathbb{E}_\pi[c(s,a)] = \sum_{s,a} c(s,a)\rho_\pi(s,a)$. It is known that there is one-to-one correspondence between the policy and its occupancy measure [43] such that $\pi(a|s)$ can be identified with $\frac{\rho(s,a)}{\sum_{a'} \rho(s,a')}$. Hence, matching the occupancy measure would imply matching the policy that corresponds to it. Utilitizing this property, many IRL approaches use a metric known as *feature expectation* that gives rise to the key equation for this paper:

$$\mu^\pi \triangleq \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \varphi(s_t, a_t)|\pi] = \mathbb{E}_\pi[\varphi(s,a)] = \mathbb{E}_{s,a\sim\rho(s,a)}[\varphi(s,a)] \tag{5.1}$$

5.2.2 Maximum Mean Discrepancy for Moment Matching

In this section, we establish the need and advantage of computing Maximum Mean Discrepancy with a characteristic kernel mean embedding [52]. By Eq. (5.1), we notice the feature expectation is indeed the mean embedding of the distribution $\rho_\pi(s,a)$ with the feature map $\varphi(s,a)$. The observation motivates the use of a kernel to represent the feature expectation.

Kernel Mean Embedding Let P, Q be Borel probability measures on $X, Y \in (S, A)$. The mean of a feature map $\mu \in \mathcal{H}$ is $\mu_p = [\dots \mathbb{E}_p[\varphi(X)] \dots]$ and for a positive definite kernel $k : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, it gives $\mathbb{E}_{P,Q}[k(x,y)] = \langle \mu_P, \mu_Q \rangle_{\mathcal{H}}$. Also we get $\mathbb{E}_p[\varphi(X)] = \langle \mu_P, \varphi(\cdot) \rangle_{\mathcal{H}}$. In relation to the main topics of the paper, the last equivalence implies the reproducing kernel property of k where choosing the kernel gives rise to an implicit feature map $\varphi : S \times A \to \mathcal{H}$ with μ_ρ^π being an

element of an RHKS $\mathcal{H}$ [54]. That is:

$$\mu_\rho \triangleq \mathbb{E}_{X\sim\rho}[k(x,\cdot)] = \mathbb{E}_{X\sim\rho}[\varphi(x)] = \int_{\mathcal{X}} \varphi(x)d\rho(x) \tag{5.2}$$

Maximum Mean Discrepancy (MMD) Let $D_x \triangleq \{x_i\}_{i:1:n}, D_y \triangleq \{y_i\}_{i:1:m}$ be the two sets of samples and suppose we would like to ask whether $P_X = P_Y$. MMD [14] is a distance measure between two distributions $P(X)$ and $Q(Y)$ defined as $\text{MMD}(P, Q, \mathcal{H}) \triangleq ||\mu_p - \mu_q||^2_{\mathcal{H}}$. We observe the inner risk in Eq. (**??**) is indeed $\text{MMD}[\rho_{\pi_\theta}, \rho_{\pi_E}, \mathbb{R}^d] = \sup_{||\varphi||\leq 1} \langle \mu^{\pi_\theta} - \mu^{\pi_E},\ \varphi(s,a)\rangle$. By Cauchy–Schwarz inequality, it is easy to notice the optimum is $||\mu^{\pi_\theta} - \mu^{\pi_E}||_2$ with $\frac{\mu^{\pi_\theta}-\mu^{\pi_E}}{||\mu^{\pi_\theta}-\mu^{\pi_E}||}$ as the maximizer. Finally, define an empirical biased estimate of MMD: $\widehat{\text{MMD}}[D_x, D_y, \mathcal{H}] = ||\frac{1}{n}\sum_{i=1}^n \varphi(x_i) - \frac{1}{m}\sum_{j=1}^m \varphi(y_j)||^2 = \frac{1}{n^2}\sum_{i=1}^n\sum_{i'=1}^n k(x_i, x_i') + \frac{1}{m^2}\sum_{j=1}^m\sum_{j'=1}^m k(y_j, y_j') - \frac{2}{nm}\sum_{i=1}^n\sum_{j=1}^m k(x_i, y_j)$.

The connection is especially useful for kernels that are injective, known as *characteristic kernels*. i.e. $||\rho_p - \rho_q||_{\mathcal{H}} = 0$ if and only if $P(X) = Q(Y)$. In relation to the main topic, this means if two policies have the same feature expectations then all of moments of the two distributions are equal. Notice we can exchange π and ρ by the one-on-one correspondence between policy and occupancy measure. This motivates the comparison of feature expectations of two policies as a metric [12].

5.3 Generative Moment Matching Imitation Learning

GMMIL (Algo. 6) directly follows from our earlier observation that occupancy measure matching induces the same optimal policy as would be given by IRL and that minimizing MMD using a characterisic kernel is essentially the same as minimizing a distance between all moments of a candidate policy and the expert policy. [25] suggested using the sum of two Gaussian kernels that has the all-moment-matching property $k(x, x') = \exp(-\frac{||x-x'||_2^2}{\sigma^2})$ to have $\rho_{\pi_\theta} = \rho_{\pi_E}$ when $\text{MMD}(\cdot) = 0$. Let $D_E = D_{\pi_E} = \{(s_j, a_j)^E\}_{j:1:m}$ and $D_\theta = D_{\pi_\theta} = \{(s_i, a_i)\}_{i:1:n}$.

Algorithm 6 Generative Moment Matching Imitation Learning [25]

Input: $\pi_\theta^{(0)}, k, D_E, \varepsilon > 0$
Parameter: θ
Output: π_θ

1: set $i = 0$
2: **while** $\widehat{MMD}[D_\theta, D_E] \leq \varepsilon$ **do**
3: Sample D_θ with $\pi_\theta^{(i)}$
4: Compute $\widehat{MMD}[D_\theta, D_E]$ for all $(s, a) \in D_E$
5: Set the optimal cost function $c^*(s, a) = \widehat{MMD}$
6: Update θ with c^* using TRPO algorithm and set $i = i + 1$
7: **end while**
8: **return** π_θ

The authors suggested the median heuristics for the bandwidth parameters $\{\sigma_1, \sigma_2\}$ where $\sigma_1 = \text{median}(\{||x_\theta^{(0)} - x_E||_2^2 \mid (x_\theta^{(0)}, x_E) \in G\})$ and $G = \{(x_\theta^{(0)}, x_E) \mid x_\theta^{(0)} \in D_\theta, x_E \in D_E\}$. For σ_2, only with D_E. Note the authors used a multi-layer neural network to represent the candidate policy as a Gaussian distribution that samples the actions according to its mean and variance that are learned. They used TRPO as a policy optimization algorithm of which we omit the details due to the space constraint [51].

5.4 Experiments

We evaluated GMMIL on 5 different environments. First, we tested the reproducibility of the algorithm on the two OpenAI control tasks for the comparison to the original paper. This was a valuable first step as the poor reproducibility is a big problem in the field of reinforcement learning [21]. Second, we designed a simple navigation task to better understand the behavior of GMMIL under the varying input dimensions and noise levels.

The baseline was a Behavior Cloning (BC) algorithm that employs supervised learning. i.e. it simply tries to learn a map $S \to A$ with either the mean squared error loss for continuous actions or the cross-entropy loss for discrete actions.

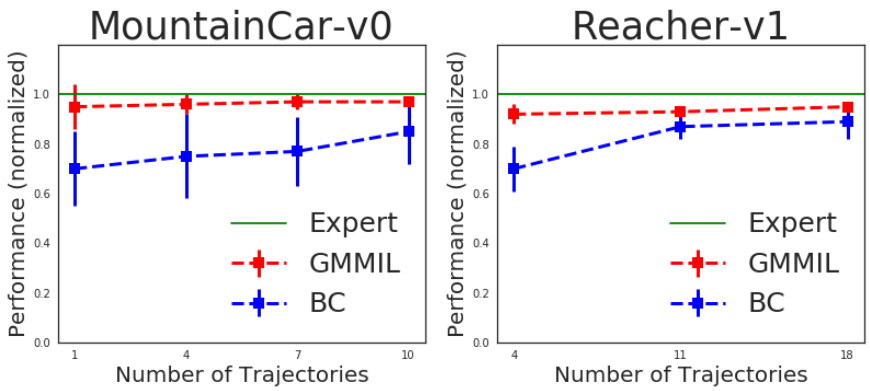

Figure 5.4.1: Reproducibility of the Algorithm on OpenAI tasks: The two plots show the normalized performance as a function of the number of training trajectories. Overall, we observed our result is on par with the original paper's result.

Since the original implementation details of GMMIL are not fully available, we based our project off the OpenAI Baseline library. Due to the computational and time constraint, our experiments were limited to 5 independent trials.

5.4.1 OpenAI Continuous-state Control Tasks

Can the results be easily reproduced? According to the reported results, the method outperformed the behavior cloning algorithm in terms of sample efficiency, total costs and stability (low variance). We chose a discrete-action problem, MountainCar-v0 ($dim(S) = 2, dim(A) = 3$), and a continuous-action problem, Reacher-v1 ($dim(S) = 11, dim(A) = 2$). The demonstration data for MountainCar-v0 was generated by the expert policy obtained with Deep Q Learning. For Reacher-v1, we used the demonstration data made available by OpenAI that is known to be obtained using TRPO [50].

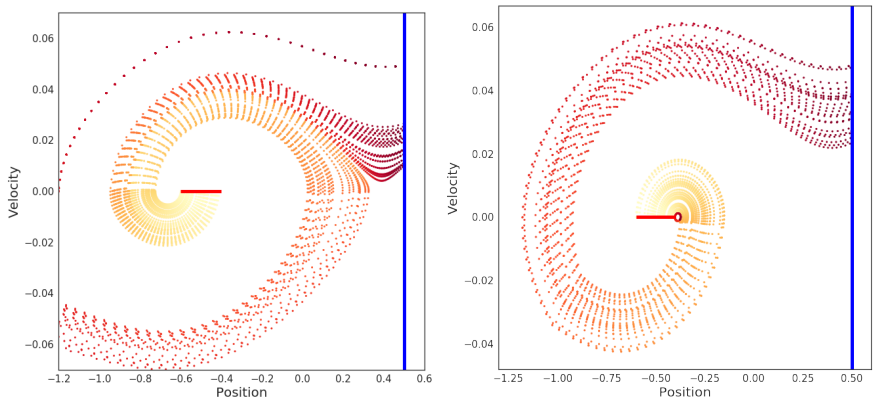

Figure 5.4.2: Trajectory Distribution of GMMIL: The two plots illustrate the trajectories (red) of the expert policy (the first) and the policy learned by GMMIL (the second). We observed the GMMIL policy learned to reach the goal (blue) but generate slightly different trajectory patterns. This is likely due to the stochasticity of the initial state distribution that pushed the agent to unseen situations.

5.4.2 Synthetic Continuous-state Navigation Task

We designed a simple continuous-state discrete-action stochastic MDP where the state is a unit hypercube $S \in [0, 1]^d$. We fixed the action set to the four directions where each action would move the agent by a constant step in the chosen direction. The agent starts at the origin and the goal is fixed on the opposite side of the diagonal which is $\sqrt{d}$ away. The dynamics is stochastic where a uniformly random action is taken instead of the action sampled by the policy with a small probability. The expert policy was hand-crafted by alternating the right and up actions towards the goal. Subsequently, the trajectories were sampled around the diagonal of the state space.

Does the performance degrade as the input dimension or the noise increases? It is known that the MMD suffers the curse of dimensionality [45]. However, the potential downside was not strongly exhibited in the test results of the original paper, even though some tasks have high dimensional inputs (e.g. 376-dimensional state variables in *Humanoid*). We hypothesized the

Dim.	GMMIL (t)	BC (t)
5	1.11 ± 0.03	1.29 ± 0.13
50	1.17 ± 0.05	1.34 ± 0.17
500	1.36 ± 0.04	1.55 ± 0.22

Noise	GMMIL (t)	BC (t)
Low	1.15 ± 0.05	1.32 ± 0.17
High	2.93 ± 0.16	3.95 ± 1.30
Nuisance	1.27 ± 0.05	1.42 ± 0.16

Table 5.4.1: Sensitivity to Input Dimension and Noise on the Synthetic task: The left table shows the behavior of GMMIL with varying input dimensions. The reported numbers refer to the average time step at which the agent reached the goal (normalized by the expert's record) with one standard error. We observed GMMIL's performance degrades as the input dimension increases but with a slower rate than BC. The right table shows the robustness to the noise. We observed that both methods are not so robust even for the low noise setting. GMMIL consistently showed a lower variance than BC.

dimensionality issue had been mitigated by the fact a subset of the high-dimensional state variables was essentially nuisance information or that the issue arises for a very large scale problem. To the end, we varied the dimension of the states $d = \{5, 50, 500\}$ and also checked how GMMIL behaves to the noise and the nuisance information. We used an isotropic Gaussian noise $\varepsilon \sim N(0, \delta^2 I)$ and varied $\delta = [0.01, 1.0]$ for $d = 50$. For the nuisance factors, we concatenated a near-constant random vector sampled independently of the states with the true state vector for $d = 5$.

5.5 Discussion and Conclusion

We empirically evaluated GMMIL to better understand its advantages and disadvantages. We reproduced its superior performance on OpenAI benchmark control tasks, compared to the supervised learning baseline (BC) and the random policy. We confirmed GMMIL is especially superior in the low training data regime which is in line with the original authors' claim that GMMIL is robust for sparse demonstrations.

In addition, we observed that GMMIL showed a low variance, making it

deemed stable. Also we observed that GMMIL is not particularly robust to the noise, but that it performed reasonably well when the nuisance information was added, which can potentially explain the GMMIL's good performance even in some high-dimensional tasks. Moreover, we observed GMMIL's performance degraded as the input dimension increased.

As for the future work, it would be interesting to study the sensitivity of GMMIL to the choice of kernels including non-characteristic and its hyper-parameters and also the compatibility with policy optimization algorithms other than TRPO. Moreover, it would be an interesting research question to adapt the algorithm to be fully applicable under the batch constraint.

Journey is the reward.

Unknown

6 Conclusion

The IRL problem is generally difficult and the batch constraint makes it even more difficult. Historically, most literature worked around the issues by making certain assumptions that may be costly in some domains (simulator and/or environment dynamics), or pursue a more simple alternative (imitation learning). There is not yet much work that discusses IRL under the batch constraint, even though the domains that impose it are increasingly interested in applying IRL.

To our knowledge, this thesis is among the first works that touch upon the topic. The TRIL+DSFN algorithm we proposed is more of a framework, than a method since it is extremely flexible. It is not coupled with any specific IRL method. It can be used with other warm-starting methods as long as they provide a good initial policy and a feature map. Our empirical evaluation showed highly promising results. Especially significant was the margin by which it

outperformed baseline methods in the Sepsis treatment task.

6.1 Future Work

We remark that there exist many open questions in the field of IRL under the batch constraint. We describe a few questions that lie on the frontiers.

- Transferability: one of the main open problems in reinforcement learning is how to learn a policy or a reward function, optimal for one task, can scale across other related tasks. It would an important contribution to evaluate the TRIL+DSFN algorithm across multiple tasks, similar to [3].
- Interpretation of reward functions: as the IRL problem is ill-posed, usually there is no guarantee of recovering the original reward function an expert is assumed to act with respect to. The TRIL+DSFN algorithm also has no guarantee. This absence of guarantee limits our hope of interpreting expert behaviors through a learned reward function. It would be a valuable extension to formulate conditions in which the recovery of expert reward functions becomes more probable, similar to [11].
- The Growing Batch Constraint: In any fields, batch data will grow over time as additional data come in. One could relax the batch constraint such that exploration via the suggestion of a behavior policy is possible at a long interval. The length of the interval and what kind of behavior policies are allowed would distinguish the growing batch constraint from the usual online learning setting.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 897–904, 2011.

[3] André Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zídek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *arXiv preprint arXiv:1901.10964*, 2019.

[4] Michael Bloem and Nicholas Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*, pages 4911–4916. IEEE, 2014.

[5] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[7] Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*, 2011.

[8] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr): 503–556, 2005.

[9] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM, 2004.

[10] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[11] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. 2017.

[12] Kenji Fukumizu, Francis R Bach, and Michael I Jordan. Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. *Journal of Machine Learning Research*, 5(Jan):73–99, 2004.

[13] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1): 1437–1480, 2015.

[14] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[15] Alexander Grubb and J Andrew Bagnell. Generalized boosting algorithms for convex optimization. *arXiv preprint arXiv:1105.2054*, 2011.

[16] Yann Guermeur. Vc theory of large margin multi-category classifiers. *Journal of Machine Learning Research*, 8(Nov):2551–2594, 2007.

[17] Michael Herman, Tobias Gindele, Jörg Wagner, Felix Schmitt, and Wolfram Burgard. Inverse reinforcement learning with simultaneous estimation of rewards and dynamics. In *Artificial Intelligence and Statistics*, pages 102–110, 2016.

[18] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. Deep q-learning from demonstrations. 2017.

[19] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[20] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[21] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[22] Ming Jin, Andreas Damianou, Pieter Abbeel, and Costas Spanos. Inverse reinforcement learning via deep gaussian process. 2015.

[23] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. volume 3, page 160035. Nature Publishing Group, 2016.

[24] Matti Kääriäinen. Lower bounds for reductions. In *Atomic Learning Workshop*, 2006.

[25] Kee-Eung Kim and Hyun Soo Park. Imitation learning via kernel mean embedding. *AAAI*, 2018.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Edouard Klein, Matthieu Geist, and Olivier Pietquin. In *European Workshop on Reinforcement Learning*, pages 285–296. Springer, 2011.

[28] Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. In *Advances in Neural Information Processing Systems*, pages 1007–1015, 2012.

[29] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. volume 4, pages 1107–1149, 2003.

[30] Michail G Lagoudakis, Ronald Parr, and Michael L Littman. Least-squares methods in reinforcement learning for control. In *Hellenic conference on artificial intelligence*, pages 249–260. Springer, 2002.

[31] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[32] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1342–1350, 2010.

[33] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1342–1350, 2010.

[34] Singer Mervyn, Deutschman Clifford S., Seymour Cristopher, and et al. The third international consensus definitions for sepsis and septic shock (sepsis-3). *JAMA*, 315(8):801–810, 2016. doi: 10.1001/jama.2016.0287.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. volume 518, page 529. Nature Publishing Group, 2015.

[37] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[38] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[39] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

[40] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128, 2017.

[41] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Learning from demonstrations: Is it worth estimating a reward function? In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 17–32. Springer, 2013.

[42] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted and reward-regularized classification for apprenticeship learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1249–1256. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[43] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[44] Aniruddh Raghu, Matthieu Komorowski, Leo Celi Ahmed, Imran, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. 2017.

[45] Aaditya Ramdas, Sashank Jakkam Reddi, Barnabás Póczos, Aarti Singh, and Larry A Wasserman. On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions. In *AAAI*, pages 3571–3577, 2015.

[46] Daniel G Remick. Pathophysiology of sepsis. *The American journal of pathology*, 170(5):1435–1444, 2007.

[47] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[48] Stéphane Ross and Joelle Pineau. Model-based bayesian reinforcement learning in large structured domains. In *Uncertainty in artificial intelligence: proceedings of the... conference. Conference on Uncertainty in Artificial Intelligence*, volume 2008, page 476. NIH Public Access, 2008.

[49] Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[50] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

[52] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *International Conference on Algorithmic Learning Theory*, pages 13–31. Springer, 2007.

[53] Zhao Song, Ronald E Parr, Xuejun Liao, and Lawrence Carin. Linear feature encoding for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4224–4232, 2016.

[54] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[55] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[56] Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

[57] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. corr (2015). *arXiv preprint cs.LG/1509.06461*, 2015.

[58] Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*, pages 6288–6297, 2018.

[59] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.