

# FRONTIERS IN INVERSE REINFORCEMENT LEARNING

SAYAN SARKAR

A thesis submitted for the partial fulfillment of the requirements for  
the BS-MS Dual Degree Programme

Department of Mathematics  
Indian Institute of Science Education and Research, Pune



*Supervisor:* Tomáš Gavenčíak  
Department of applied mathematics, Charles University, Prague

October 2019

Sayan Sarkar : *Frontiers in Inverse Reinforcement Learning*

© October 2019

This thesis is published under a

Creative Commons Attribution-NonCommercial 4.0 International  
License.

The entire license can be found at

<https://creativecommons.org/licenses/by-nc/4.0/>.

## CERTIFICATE

---

This is to certify that this dissertation entitled *Frontiers in Inverse Reinforcement Learning* towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Sayan Sarkar jointly at Indian Institute of Science Education and Research, Pune and Charles University, Prague under the supervision of Tomáš Gavenčík, Department of Applied Mathematics, Charles University, during the academic year 2019.



---

Tomáš Gavenčík

## COMMITTEE

---

Tomáš Gavenčík  
Anup Biswas



Sayan Sarkar

“We understand the lights. We understand the lights above the Arby’s. We understand so much. But the sky behind those lights, mostly void, partially stars, that sky reminds us: we *don’t* understand even more.”

Welcome to Night Vale  
Episode 25 – One Year Later

Dedicated to the human quest for knowledge.

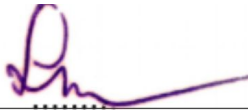
## DECLARATION

---

I hereby declare that the matter embodied in the thesis entitled **FRONTIERS IN INVERSE REINFORCEMENT LEARNING** is the result of the work carried out by me at *Department of Mathematics, Indian Institute of Science Education and Research, Pune* and *Department of Applied Mathematics, Charles University, Prague* under the supervision of Tomáš Gavenčík and the same has not been submitted elsewhere for any other degree.

*Pune, India, October 2019*

  
Sayan Sarkar



---

Tomáš Gavenčík

## ABSTRACT

---

How do we *teach* machines to do something that we can perform reasonably well, but cannot easily express as a *utility maximization problem*? Can machines learn underlying utility of a domain from many *human demonstrations*?

The goal of the field of [Inverse Reinforcement Learning \(IRL\)](#) is to infer the crux of the *goal* of a domain from expert (human) demonstrations. This thesis categorically surveys the current IRL literature with a formal introduction and motivation for the problem. We discuss the central challenges of the domain and expound upon how different algorithms deal with the challenges. We propose a reformulation of the [IRL](#) problem by including a ranked set of trajectories of different levels of expert capability and discuss how that might lead towards a new set of algorithms in the field, motivated by some very recently developed approaches. We conclude with discussing some broad advances in the research area and possibilities for further extension.

## PUBLICATIONS

---

Some ideas and figures have appeared previously in the following publications:

- [1] Adria Garriga Alonso, Max Marian Daniel, Johannes Heidecke, Anton Osika, Sayan Sarkar. *IRL Benchmark - A Python Package for Benchmarking Inverse Reinforcement Learning Algorithms*. URL: <https://github.com/JohannesHeidecke/irl-benchmark>.
- [2] Arushi Majha, Sayan Sarkar, and Davide Zagami. "Categorizing Wireheading in Partially Embedded Agents." In: *Artificial Intelligence Safety 2019, AISafety@IJCAI 2019, Macao, China, August 11-12, 2019*. 2019. URL: [http://ceur-ws.org/Vol-2419/paper%5C\\_31.pdf](http://ceur-ws.org/Vol-2419/paper%5C_31.pdf).

*Friends are those rare people who ask how we are,  
and then wait to hear the answer.*

— Ed Cunningham

## ACKNOWLEDGMENTS

---

I am immensely grateful to the continued support, both academic and personal, of my thesis supervisor Tomáš Gavenčíak throughout the project. I could not have completed the project without the regular thoughtful discussions and pithy comments from him. My thesis committee member Dr. Anup Biswas has helped me extensively to align and ground the thesis in both practicality and mathematical rigor.

Discussions with many people have helped me deconfuse myself about the domain. Many thanks go to Johannes Heidecke, Adria Garriga Alonso, Max Marian Daniel, Anton Osika, Imran Rashid, Jean Garcin, Darlene Moss, Davide Zagami, and Arushi Majha. I extend my gratitude to my friends who have been there for me in times good and bad.

The online communities at the subreddit `/r/machinelearning` and `/r/reinforcementlearning` has helped me extensively to understand the entire discipline of machine learning. I am indebted to Debarshi Mitra for helping with proofreading and editing this document.

Finally, special thanks to Anindita Basu and Mr. Gollu.



# CONTENTS

---

1	INTRODUCTION	17
1.1	Goal and Scope	17
1.2	Contributions	17
1.3	Overview	18
2	PRELIMINARIES	19
2.1	Reinforcement Learning	19
2.2	Motivation	19
2.3	Reinforcement Learning Formalisation	20
2.3.1	Agent and Environment	20
2.3.2	Markov Decision Processes	21
2.3.3	Policy and Value Functions	22
2.3.4	Bellman Equations	23
2.3.5	Solving an MDP	25
3	INVERSE REINFORCEMENT LEARNING	27
3.1	Motivation	27
3.2	Problem Statement	28
3.3	Importance of IRL	28
3.4	Related Areas	29
3.5	Challenges	29
3.5.1	Underspecification	29
3.5.2	Generalizability	31
3.5.3	Correctness of Prior Knowledge	31
3.5.4	Solution Complexity	31
4	CLASSICAL IRL ALGORITHMS	33
4.1	Linear Programming IRL from Trajectories	33
4.2	Feature Matching	35
4.2.1	Featurization	36
4.2.2	Feature Matching	36
4.3	Support Vector Machines Formulation	37
4.4	Maximum Entropy Methods	37
4.4.1	Rationality Model	38
4.4.2	Boltzmann Rationality	38
4.4.3	Guided Cost Learning Formulation of Maximum Entropy IRL	39
4.4.4	Maximum Causal Entropy Methods	40
4.5	Bayesian Methods	41
4.5.1	Bayesian Inverse Reinforcement Learning	41
5	MODERN IRL ALGORITHMS	45
5.1	Deep Learning and IRL	45

5.2	GANs and IRL . . . . .	46
5.3	Adversarial Methods . . . . .	47
5.3.1	Empowerment Based Adversarial Inverse Reinforcement Learning . . . . .	47
6	TOWARDS NEW IRL ALGORITHMS	51
6.1	Inverse Reinforcement Learning with Ranked Tra- jectories . . . . .	51
6.1.1	Related Work . . . . .	51
6.1.2	Definitions . . . . .	53
6.1.3	Algorithm 1: Rank Difference as Reward Function Proxy . . . . .	55
6.1.4	Algorithm 2: Bayesian RWSD Matching . .	55
6.1.5	Experiments . . . . .	56
6.1.6	Comments . . . . .	59
6.2	Conditional GANs and Ranked IRL . . . . .	59
7	CONCLUSIONS	63
7.1	Discussions . . . . .	63
7.1.1	Breakthroughs . . . . .	63
7.1.2	Shortcomings . . . . .	64
7.2	Future Work . . . . .	64
	<b>Appendix</b>	
	Acronyms	73

## LIST OF FIGURES

---

Figure 2.1	The Reinforcement Learning Setup. Image by Lilian Weng. . . . .	20
Figure 2.2	A graphical representation of an MDP. States are denoted by hexagones; actions by circles, transition probabilities by green arrows and action selections by orange arrows. . . . .	22
Figure 3.1	The standard IRL problem. Reused under a Creative Commons Attribution License from [43] . . . . .	28
Figure 4.1	A sample five-state MDP. Image courtesy of the MAP-BIRL paper. . . . .	43
Figure 4.2	The corresponding recovered reward posterior to the five-state MDP as obtained by a MAP-BIRL algorithm. Image courtesy of the MAP-BIRL paper. . . . .	43
Figure 5.1	The Deep MEIRL pipeline. Image obtained from the original paper. . . . .	46
Figure 6.1	Violinplot for the reward function obtained in the 40 runs of the algorithm. . .	58
Figure 6.2	Boxlplot of <i>only</i> the low reward values of the frozen states vs all the zero reward values of the hole cells. Frozen states correspond to 0 and holes correspond to 1. . . . .	58
Figure 6.3	A comparison of the architectures of GANs and Conditional GANs. Image courtesy: Lilian Weng. . . . .	61



## LIST OF TABLES

---

Table 4.1	Various IRL algorithms as specific cases of the MAP-BIRL framework. . . . .	44
Table 5.1	Comparison of various IRL algorithms in terms of key properties . . . . .	49



## LIST OF ALGORITHMS

---

Figure 1	Sarsamax (Q-Learning) . . . . .	26
Figure 2	IRL from Sampled Trajectories . . . . .	35
Figure 3	Maximum Margin Separating Hyperplane Algorithm . . . . .	37
Figure 4	EAIRL . . . . .	48





## INTRODUCTION

---

This master's thesis project is an overview of the nascent field of [Inverse Reinforcement Learning \(IRL\)](#) - the algorithms, challenges, and applications. In the past two decades or so, [IRL](#) has been used in diverse disciplines of artificial intelligence, psychology, control theory, and machine learning. The appeal of the field lies in the potential to use data recorded in a task to build autonomous agents able to model other agents without intervening in the continued performance of the task.

### 1.1 GOAL AND SCOPE

The goal of this project is two-fold. Firstly, this project introduces and discusses the history of the field succinctly with a birds-eye-view. Secondly, this thesis discusses paths forward to novel [IRL](#) algorithms with a reformulation of the problem.

The scope of this project is both practical and theoretical - we discuss and develop the intuitions and mathematics behind the subject. We provide adequate proof and reasoning behind the major milestones. Apart from that, we provide a full reference implementation of a canonical algorithm.

### 1.2 CONTRIBUTIONS

This thesis makes several contributions that are summarized below.

- Formally introducing the [IRL](#) problem and comparison of the available methods with several examples.
- An investigation of the main difficulties that arise in the paradigm of [IRL](#) and a systematic review of the partial solutions offered so far.
- Discerning the key milestones and common shortcomings.
- Developing open avenues for future research with novel [IRL](#) methodologies alongside a reformulation of the problem itself.

## 1.3 OVERVIEW

This thesis is organized in an almost linear and chronological fashion. The ordering of the chapters facilitates a reading that unfolds the field historically but also provides the challenges and motivations beforehand.

In chapter 2 we introduce the mathematical background necessary for developing IRL. This includes the sufficient statistical tooling and an ample description of the RL problem - the *forward* counterpart of the IRL problem.

In chapter 3 we formally introduce the IRL problem and delineate the main challenges associated with it.

Chapter 4 discusses the old algorithms from the *classical* era of IRL - before the dawn of the success of the modern Deep Learning. These algorithms, though inefficient and fragile in hindsight, provided the bedrock of the discipline.

Chapter 5 introduces the *modern* algorithms that are almost all dependent on the gigantic success of Deep Learning (DL). We also briefly discuss Generative Adversarial Network (GAN)s and develop the analogy with IRL.

Chapter 6 is a collection of work-in-progress IRL algorithms and the motivations behind them. These algorithms are mostly built around and similar to the recently developed approach of using multiple expert demonstrations of variable optimality such as the T-REX method [9]. Most of the chapter is in an inchoate form and only a direction towards full-brown algorithms.

In chapter 7, we conclude the thesis with discussions and future work.

In the appendix, we present a reference implementation of one of the IRL algorithms in Python.

*That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).*

---

Richard Sutton

## 2.1 REINFORCEMENT LEARNING

**Reinforcement Learning (RL)**, along with *Supervised* and *Unsupervised Learning*, is a major branch of contemporary machine learning approaches. Instead of giving the agent labeled or unlabeled data points to learn from, the **RL** approach uses a *reward signal*, which the agent must cleverly use to achieve its goals in the environment, whatever that might be. **RL** differs from traditional *supervised* and *unsupervised learning* paradigms in several aspects. Firstly, in **RL**, there is no supervisor, that provides information about how 'successful' the learning is. Secondly, feedback is not instantaneous, but rather delayed. Thirdly, **RL** deals with non-*i.i.d.* data, as the *observation* of the agent depends on its *actions*. All these differences make **RL** an interesting and challenging subfield of study. However, in recent times, **RL** has been successfully applied to problems like Go, Chess, and Shogi [44] ; Protein Folding [42]; Graph Coloring [24], Atari video games [35] etc.

## 2.2 MOTIVATION

The idea of Reinforcement Learning is to formalize and to learn by *trial-and-error*. Rather than explicitly assigning to the agent what the optimal behavior is, this framework allows the agent to experiment by itself, and learn from the environment, not much unlike how human children learn. [47]

In any **RL** problem, we have an *agent*, which exerts control over its future by making decisions. The agent acts inside an

**environment** that it can interact with, but cannot control. The agent usually has a **goal** of achieving something - which is formalized by changing the *state* of the environment. The **learning** happens when the agent figures out a strategy to achieve its goals with high confidence.

RL achieves the goal, or 'learns' the solution, by defining a scalar feedback signal called the *reward*<sup>1</sup> that quantifies how well the agent is doing at a particular step at achieving a specific task. The agent's job is to maximize *cumulative total reward*. The entire field of RL is based on the following hypothesis:

**REWARD HYPOTHESIS:** All goals can be described by the maximization of expected cumulative reward.

Various real-world problems are easy to model with the reward framework. For example, in case of learning the game of Chess, we might design the environment in a way that the agent receives positive rewards for winning the game and a negative reward for losing it. In this manner, the reward framework makes itself conducive to learning in complex domains where the goal is easy to specify but a solution is pretty convoluted to write down via expert systems.

<sup>1</sup> some related fields use the notion of *cost* which is just the negative of the reward and conceptually similar. However, another similar concept is that of reward, but we should be careful to not conflate these two. [25]

## 2.3 REINFORCEMENT LEARNING FORMALISATION

### 2.3.1 Agent and Environment

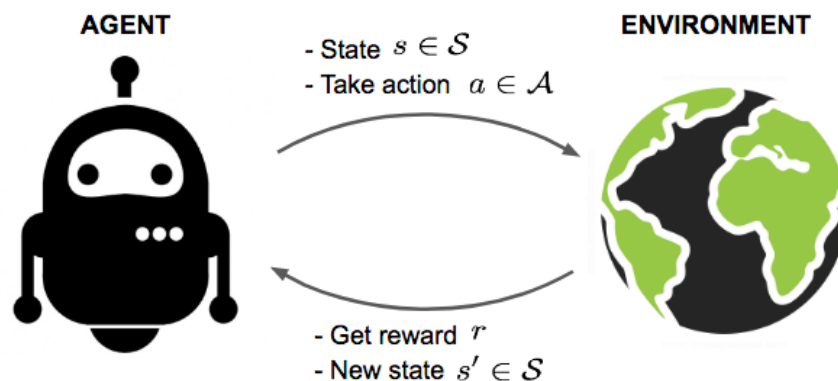


Figure 2.1: The Reinforcement Learning Setup. Image by Lilian Weng.

In an informal sense, the **agent** is something that exerts control over the **environment**. At any time-step, the agent has

information about itself and the environment, which we call **observations**. Observation may not capture everything about the environment - we cannot expect the agent to know everything about the world. The environment usually consists of a continuous or discrete **state-space**. The agent, at any step, can usually choose an action from the **action-space**, which can again be discrete or continuous. At each step, once the agent chooses an action, the environment provides a numerical feedback signal, the **reward**. Also, the state of the environment changes according to the **transition dynamics function**, given the current state and the chosen action.

We capture all of this information into a framework called **Markov Decision Process (MDP)**.

### 2.3.2 Markov Decision Processes

#### Definition 1: Markov Decision Process

A Markov Decision Process is a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ .

- $\mathcal{S}$  is the discrete or continuous state-space.
- $\mathcal{A}$  is the discrete or continuous action-space, available for each state.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, \infty]$  is the reward function.  $\mathcal{R}(s, a, s')$  determines the numerical reward obtained when taking a particular action  $a$  in a particular state  $s$  which leads the agent to the next state  $s'$ .
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  is the transition probability. If the state and actions are discrete, this can be represented as a tensor.  $\mathcal{P}(s, a, s')$  determines the probability of reaching state  $s'$ , when taken action  $a$  in state  $s$ .
- $\gamma \in [0, 1]$  is the discount factor, which quantifies the value of future rewards in current time-step.

We can equivalently define the reward function to be state only as  $\mathcal{R}(s)$  providing the reward obtained for *being* in state  $s$ , or state-action only as  $\mathcal{R}(s, a)$  codifying the reward obtained when taking action  $a$  in state  $s$ .

It is easy to note that both the transition probability and the reward function depend only on the current state, and the action, hence it respects the *Markov Property*.

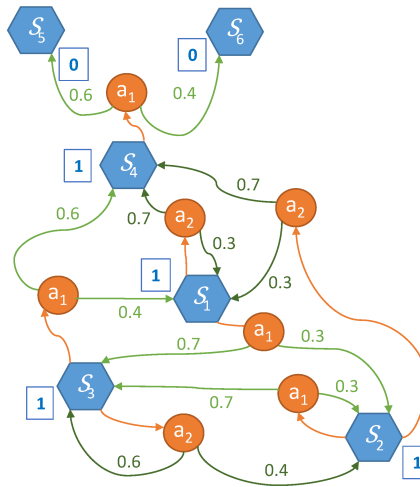


Figure 2.2: A graphical representation of an MDP. States are denoted by hexagones; actions by circles, transition probabilities by green arrows and action selections by orange arrows.

### 2.3.3 Policy and Value Functions

A **policy** is a mapping from states to actions that guide the agent to take actions while in a state. The agent chooses which action to take in what state based on a **deterministic** or **stochastic** policy function. The goal of Reinforcement Learning is to make the agent come up with an optimal policy by interaction with the environment.

It is often useful to use a **discount factor**,  $\gamma$ , to simplify the summation of infinite series in value functions. It quantifies the *value of future rewards in the current time-step*. There are several reasons to use the discount factor, which we will not discuss here.

A useful tool for arriving at a policy is to attach some numerical 'value' to each state, or state-action pairs. In this way, the agent can quantify which states are desirable or what action at which state is better than others. An intuitive way of defining the value of a state or state-action pair is to calculate the **expected cumulative reward**. In a reinforcement learning setup, we define the cumulative reward as **total return of an episode**  $G_T = \sum_{k=1}^T \gamma^{k-1} R(s_k)$  and we define the value functions of a policy  $\pi$  as:

**Definition 2: Value Functions**

The state-only value function in an MDP with  $r_i = R(s_i)$  for state  $s$  while following policy  $\pi$  is defined as

$$v_\pi(s) = \mathbb{E}\left[\sum_{i=t}^T \gamma^{i-1} r_i\right] \quad (2.1)$$

The state-action value function in an MDP for state  $s$  and action  $a$  while following policy  $\pi$  is defined as

$$q_\pi(s, a) = \mathbb{E}\left[\sum_{i=t}^T \gamma^{i-1} r_i \mid a_t = a\right] \quad (2.2)$$

**Definition 3: Optimal Value Functions**

In an MDP, under a policy  $\pi$  the optimal state-value function is defined as

$$v_*(s) \doteq \max_{\pi} v_\pi(s) \quad \forall s \in \mathcal{S} \quad (2.3)$$

In an MDP, under a policy  $\pi$  the optimal action-value function is defined as

$$q_*(s, a) \doteq \max_{\pi} q_\pi(s, a) \quad \forall s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s) \quad (2.4)$$

2.3.4 *Bellman Equations*

Unless the agent has taken every possible action in every possible state, it does not know the rewards or the value functions. It turns out there's an efficient way to estimate and update the value function recursively. These are known as the Bellman Equations [7].

Under a policy  $\pi$ , with transition function  $P$ , value of the current step depends on the next step  $s'$  like this:

**Theorem 1: Bellman Expectation Equations**

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s, a, s') (r(s') + \gamma v_\pi(s'))$$

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} p(s, a, s') (r(s') + \gamma \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q_\pi(s', a'))$$

Similarly, we can derive the Bellman equations for the optimal value functions.

**Theorem 2: Bellman Optimality Equations**

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s, a, s') (r(s') + \gamma v_*(s'))$$

$$q_*(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') (r + \gamma \max_{a' \in \mathcal{A}(s')} q_*(s', a'))$$

We provide a proof of one of the Bellman equations 2.3.4 for completeness. Proving the other equations are mostly similar.

$G_t$  is the total return starting from time  $t$ .  $G_t = \sum_{k=t}^{\infty} \gamma^{k-t} R(s_k)$

**Note:** Convergence of the sum depends on whether the **MDP** is episodic or infinite, and whether  $\gamma$  is  $< 1$  or not. In most RL setups, there is usually some way to convert an infinite MDP into an episodic one, so this doesn't prevent us from using undiscounted ( $\gamma = 1$ ) MDPs.



*Proof.*

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \quad (1)$$

$$= \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a) \mathbb{E}_\pi[G_t | S_t = s, A_t = a, S_{t+1} = s'] \quad (2)$$

$$= \sum_{s' \in \mathcal{S}} p(s, a, s') \mathbb{E}_\pi[G_t | S_t = s, A_t = a, S_{t+1} = s'] \quad (3)$$

$$= \sum_{s' \in \mathcal{S}} p(s, a, s') \mathbb{E}_\pi[G_t | S_{t+1} = s'] \quad (4)$$

$$= \sum_{s' \in \mathcal{S}} p(s, a, s') \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_{t+1} = s'] \quad (5)$$

$$= \sum_{s' \in \mathcal{S}} p(s, a, s') (r(s') + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']) \quad (6)$$

$$= \sum_{s' \in \mathcal{S}} p(s, a, s') (r(s') + \gamma v_\pi(s')) \quad (7)$$

The reasoning for the above is as follows:

- (1) by definition ( $q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$ )
- (2) Law of Total Expectation
- (3) by definition ( $p(s, a, s') \doteq \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$ )
- (4)  $\mathbb{E}_\pi[G_t | S_t = s, A_t = a, S_{t+1} = s'] = \mathbb{E}_\pi[G_t | S_{t+1} = s']$
- (5)  $G_t = R_{t+1} + \gamma G_{t+1}$
- (6) Linearity of Expectation
- (7)  $v_\pi(s') = \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']$

□

### 2.3.5 Solving an MDP

If we can come up with a policy that takes actions that maximizes the value function, then say that the MDP is solved, and *learning* is done. A simple way to do that would be to calculate

the state-action value function is each step and then choose the action with maximum q-value.

$$\pi(a|s) = \operatorname{argmax}_{a \in \mathcal{A}} q(s, a) \quad (2.5)$$

The entire field of RL deals with estimating and evaluating these policies and value functions in an efficient manner, with a plethora of algorithms.

For reference purposes, we provide a standard RL algorithm known as Q-learning or Sarsamax [47].

---

**Algorithm 1** *Sarsamax (Q-Learning)*

---

policy  $\pi$ , positive integer  $num\_episodes$ , small positive fraction  $\alpha$ , GLIE  $\{\epsilon_i\}$  value function  $Q$  ( $\approx q_\pi$  if  $num\_episodes$  is large enough) Initialize  $Q$  arbitrarily (e.g.,  $Q(s, a) = 0$  for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}(s)$ , and  $Q(\text{terminal-state}, \cdot) = 0$ )

$i \leftarrow 1$  **to**  $num\_episodes$   $\epsilon \leftarrow \epsilon_i$

Observe  $S_0$

$t \leftarrow 0$

$S_t$  is terminal Choose action  $A_t$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Take action  $A_t$  and observe  $R_{t+1}, S_{t+1}$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$

$t \leftarrow t + 1$   $Q$

---

*After all, the entire field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy, is the most succinct, robust, and transferable definition of the task.*

---

Andrew Ng & Stuart Russell

### 3.1 MOTIVATION

Reinforcement Learning has seen some astonishing success in recent years in domains where the reward function is easy to obtain, for example, games like Chess and Go. What happens when we do not have the reward function in an explicit manner but have some vague understanding of it? What if we have access to demonstrations of what we may call desirable examples, and want our agent to learn the reward function itself? In the real-world, reward functions are not easy to obtain [27]. For example, a human driver optimizes between safety, speed, comfort, fuel usage, traffic laws, and various other aspects of everyday commute. It is enormously challenging to hand-design a *reward function* that takes into account all of it in a robust manner.

The goal of **Inverse Reinforcement Learning** is to infer the reward function from expert demonstrations or policy. The idea is to show the agent a bunch of demonstrations of how an *expert* in the real world solves a certain problem, so that it learns what problem the expert is trying to solve. The catch is that by doing so we withdraw ourselves from the messy task of specifying the exact goal in math.

One can imagine, if we can solve the problem of **IRL** efficiently, we can teach an AI even extremely complex *reward functions* such as human value systems, without explicitly hand-designing rules (which immediately results into several goal-misspecification or reward-hacking problems [16, 34]). This

is the open problem of value-learning using [IRL](#), sometimes known as *ambitious value learning* [17].

### 3.2 PROBLEM STATEMENT

We consider a *Markov Decision Process without Rewards*, **MDP/R**, which is the tuple  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma \rangle$ . Along with that, we assume that we have access to a set of **expert demonstration trajectories** in the form of **state-action pairs**  $(s_t, a_t)$ . We denote each trajectory as  $\tau_1 = (s_1, a_1), \tau = (s_2, a_2), \dots, \tau = (s_1, a_T)$ .

One fundamental assumption in early IRL algorithms is that the expert is entirely **rational**, and the trajectories are generated from an *optimal policy*. As we shall see, this is a strong assumption in the setting of the real world.

The goal of IRL is to estimate a reward function  $R$ , which explains these trajectories as generated from following an optimal policy. More formally, we are interested in finding a reward function under which the *optimal policy* in the MDP results into the expert trajectories.

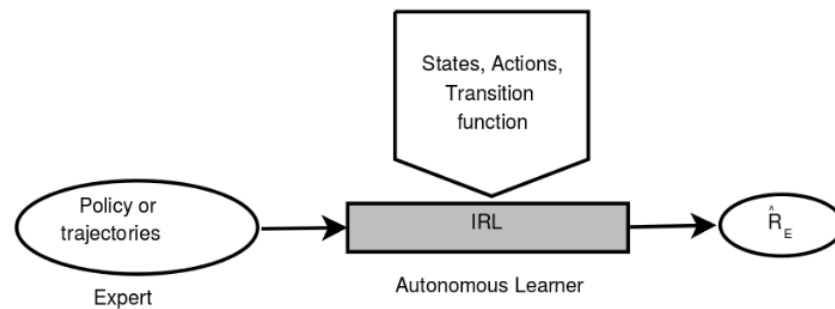


Figure 3.1: The standard IRL problem. Reused under a Creative Commons Attribution License from [43]

### 3.3 IMPORTANCE OF IRL

The field’s co-founder Stuart Russell’s original idea behind introducing [IRL](#) was to computationally model human and animal behavior to mitigate the problems in supplying those. In the recent past, there have been interesting progress in this direction. For example, [Inverse Reinforcement Learning](#) (IRL) has recently been used to model theory of mind [29].

Two broad areas of application of [IRL](#) are

- Learning from expert to create an agent with the expert’s preferences. An early and well-known application that brought attention to IRL is helicopter flight control [14].
- Learning from another agent to predict its behavior. One of the first attempts in this direction was route prediction for taxis [54]. Other applications include footstep prediction for planning legged locomotion [55].

### 3.4 RELATED AREAS

**Inverse Reinforcement Learning (IRL)** is a specific sub-domain in the field of *Learning from Demonstrations (LfD)* - where the goal of the agent is to learn some function (for example, the reward function) from expert demonstrations [5].

One of the closely related areas in LfD is *Imitation Learning* - where the agent tries to infer the **expert policy** given the expert trajectories. Recent success in autonomous vehicles [28] has highly depended on methods from Imitation Learning. However, IRL is a much more challenging task than Imitation Learning, as noted in the following sections. Besides, IRL is much more robust to distributional shift than Imitation Learning, given the portable nature of the reward function.

### 3.5 CHALLENGES

#### 3.5.1 Underspecification

##### *Representational and Experimental*

The quintessential challenge in the domain of **IRL** is the underspecified nature of the problem itself. As identified by Abbeel and Ng [1], many reward functions can explain the same policy, and many policies may lead to the same trajectories. Hence, it is theoretically impossible, to come up with a unique reward function that explains the behavior of the expert. Amin and Singh [4] classifies two kinds of unidentifiability of reward functions. It is easy to show that the optimal policy is invariant under any affine transformation of the reward function.

Hence,

$$R \equiv \lambda R + c, \text{ for } \lambda, c \in \mathbb{R}.$$

This leads to the **representational** unidentifiability - the found reward function may look totally different. This challenge has an easy solution - normalize all reward functions to  $[0, 1]$  with

the following transformation so that it is feasible to compare two seemingly different reward functions.

$$R \longrightarrow \frac{R - \max R}{\max R - \min R} \quad (3.1)$$

The second kind of unidentifiability is harder to tackle. The same trajectories can imply various policies. For example, consider a small grid world where the expert has followed a particular path. We note that it has traversed a particular cell before the final one. It is impossible to tell whether this is because this cell has some rewards or if this is simply because the agent just traversed it on its way to the final cell. This kind of ambiguity is **experimental** - it is possible to perform enough experiments and extract more information, in order to mitigate the degeneracy.

We need to change the *transition probability*  $P$  and create a set of MDPs with everything else being the same. For example, if we artificially introduce a 'wall' around the final cell in the grid world example, we will be able to know whether our cell under consideration actually contains any reward.

Though formally sounding plausible, this is impractical in real-world settings. We cannot expect to arbitrarily change the MDP or perform multiple experiments with the expert. Even if possible, this becomes enormously computationally expensive, with the introduction of multiple different subroutines.

### *Insufficiency of Occam's Razor*

A very recent paper [6] has identified a third kind of reward function, which is fundamentally much more challenging, as it comes in the form of a No-Free Lunch theorem. In real life, we cannot assume complete rationality of the expert agent, as multiple research has shown that humans are systematically and predictably irrational [49]. Previously, this has been dealt with proxies like **Boltzmann Rationality** [41] and **Occam's Razor** [26] (we will discuss the algorithms in the next sections). However, Armstrong and Minderman showed that it is impossible to infer both the 'planning algorithm' and reward function from data from irrational experts, with the help of **Occam's Razor** or the **Solomonoff Prior** [46].

### 3.5.2 *Generalizability*

Though the reward function is the most ‘succinct, robust, and transferable’ [47] definition of the task or the goal, there is still no guarantee of the reward function generalizing in an unknown or bigger environment. This problem is closely related to the problem of generalization in Reinforcement Learning, and one can expect that tools from recent research efforts in One-Shot RL or Meta RL can be applied back to IRL problems.

### 3.5.3 *Correctness of Prior Knowledge*

Most IRL algorithms compress the large state-space information into much smaller tractable ‘feature space’ via the approach of ‘feature engineering.’ For most of history, these features are hand-designed and thus prone to human bias and insufficiency. It turns out that most IRL algorithms are sensitive to the correctness of this **prior knowledge** about the environment. One way to address this problem is to form a completely end-to-end IRL solution with the help of modern deep neural networks so that the so-called feature functions are automatically computed as part of the algorithm black box from the complete state-space information.

### 3.5.4 *Solution Complexity*

Until very recently, all IRL algorithms relied on solving an entire MDP using some RL algorithm in the inner loop of its iteration. This is computationally super expensive and almost untractable in the case of real-world state spaces. Very recent methods like **Guided Cost Learning** has proposed a possible solution to this challenge by designing the algorithm in a fundamentally different way - it solves the MDP only once.





*It appears to be a quite general principle that, whenever there is a randomized way of doing something, then there is a nonrandomized way that delivers better performance but requires more thought.*

E T Jaynes

#### 4.1 LINEAR PROGRAMMING IRL FROM TRAJECTORIES

The first concrete method for IRL was introduced by Abbeel and Ng [2] in 2004 in three different contexts - small state space with access to expert policy, large/ infinite state space with access to expert policy, and with access to expert trajectory only. Abbeel and Ng formally derived the criteria for solving an IRL problem as well. The first two approaches are almost trivial in hindsight. We will go through the theorem for IRL and also discuss the third non-trivial algorithm.

##### Theorem 1: Reward Satisfiability Region in IRL

With access to the expert policy and state transition probability, the solution set for the IRL problem is any reward function that satisfies

$$\forall P^i \in P^{-\pi_E} : (P^{\pi_E} - P^i)(I_n - \gamma P^{\pi_E})^{-1} R \succeq 0 \quad (4.1)$$

where  $\succeq$  denotes the inequality  $x \succeq y \iff \forall i : x_i \geq y_i$ .

*Proof.* To prove the theorem, we start with the idea that the expert's observed policy must be optimal under the estimated reward.

$$\forall s : \pi_E(s) = \operatorname{argmax}_a \sum_{s' \in \mathcal{S}} P(s, a, s') V^{\pi_E}(s') \quad (4.2)$$

which is same as

$$\forall s, a : \sum_{s' \in \mathcal{S}} P(s, \pi_E(s), s') V^{\pi_E}(s') \geq \sum_{s' \in \mathcal{S}} P(s, a, s') V^{\pi_E}(s') \quad (4.3)$$

We can rewrite this in more compressed vector notation. Let  $P^{\pi_E}$  be the policy transition matrix of size  $n \times n$  containing the state transition probabilities for choosing  $\pi_E(s)$  in all states.

For each state, there are  $k - 1$  actions that are not chosen by the expert. We can construct  $k - 1$  policy transition matrix like above and have the set  $P^{\neg\pi_E} = \{P^1, \dots, P^{k-1}\}$ . Estimated state-value functions and estimated reward functions can also be expressed as vectors.

Let  $\succeq$  denote the inequality -  $x \succeq y \iff \forall i : x_i \geq y_i$ . Then

$$V^{\pi_E} = R + \gamma P^{\pi_E} V^{\pi_E} \iff V^{\pi_E} = (I_n - \gamma P^{\pi_E})^{-1} R \quad (4.4)$$

and

$$\forall P_i \in P^{\neg\pi_E} : P_{\pi_E} V^{\pi_E} \succeq P_i V^{\pi_E} \quad (4.5)$$

Finally leading to

$$\forall P^i \in P^{\neg\pi_E} : (P^{\pi_E} - P^i)(I_n - \gamma P^{\pi_E})^{-1} R \succeq 0 \quad (4.6)$$

□

Ng and Russell's third algorithm 2 from the paper is the first non-trivial algorithm in the field. This algorithm does not explicitly require access to the expert policy and can work with trajectories only. The algorithm starts with a random reward function and random a policy. We can write the reward function as a dot product of parameters, and formulate the value function of a trajectory in terms of these parameters. Our goal is to estimate these parameters. To do that, we generate trajectories based on the current policy, and then calculate the coefficients

of the value of the trajectory based current reward function estimate. Finally, we solve the linear programming problem to find the new reward parameters.

$\lambda$  is a hyperparameter, Russell-Ng chose it to be 2.

---

**Algorithm 2** *IRL from Sampled Trajectories*

---

1. Initialize random reward function  $R(s) = w \cdot \phi(s)$  and random policy  $\pi$ .
2. Generate trajectories based on current policy and calculate the value of the policy based on the current reward function estimate.
3. Until converged.
  - a) Using Linear Programming, maximize for all policies

$$\sum_{i=1}^k p(v^{\pi^*}(s_0) - v^{\pi_i}(s_0)) \quad (4.7)$$

such that  $|\alpha_i| \leq 1$ .

$$p(x) = \begin{cases} x & \text{if } x \geq 0 \\ \lambda x & \text{otherwise} \end{cases} \quad (4.8)$$

- b) Return new reward function.
    - c) Calculate optimal policy wrt the new reward add that to the list of policies
  4. Return final reward function.
- 

We get a new policy that is optimal wrt this new reward function and add that to the list of policies. We repeat this procedure multiple times until some criterion is satisfied and return the parameters.

## 4.2 FEATURE MATCHING

The idea of *feature matching*, originally borrowed from the fields of early supervised learning, can be used to simplify the IRL process, as detailed in the next few subsections. Rather than dealing with huge state or state-action spaces, we can extract

the essential 'features' of the space. Similar to calculating the state-visitation frequency of the expert, we can calculate which features are collected by the expert - feature expectation. We can then try to come up with a reward function, and an optimal policy that 'matches' this *expert feature expectation*.

#### 4.2.1 Featurization

We compress the state space (small, large, or infinite) into a vector  $\phi(s) \in [0, 1]^k$  of  $k$  feature functions  $\phi_i : \mathcal{S} \rightarrow [0, 1]$ . If the reward is linear in features, we write it as

$$R(s) = w \cdot \phi(s) = \sum_{i=1}^k w_i \cdot \phi_i(s)$$

#### 4.2.2 Feature Matching

Assume that the true reward is bounded in  $[-1, 1]$  and can be expressed as

$$R^*(s) = w^* \cdot \phi(s)$$

Total Expected Value of a policy is

$$V^\pi(s_0) = w \cdot \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right] \quad (4.9)$$

Define, for each feature, the function

$$\mu_i(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right]$$

and then write  $\mu(\pi)$  as the vector of all  $\mu_i(\pi)$ , and call this vector **feature expectation of  $\pi$**

The value of a policy is

$$V^\pi(s_0) = w \cdot \mu(\pi)$$

Given that the value of expert policy is more than any other policy

$$V^{\pi_E}(s_0) \geq V^\pi(s_0)$$

which is equivalent to

$$w \cdot (\mu_E - \mu(\pi)) \geq 0$$

If trajectory, instead of policy, is available, we can easily estimate the **empirical feature expectation** from the **empirical return**.

Feature matching is simply the idea that the algorithm should learn such a representation of the reward function that the optimal policy under it *matches* the expert's empirical feature expectation.

Many algorithms have been developed with the idea of matching features. We discuss one that is one of the earliest ones, and use Support Vector Machines [48] as the core engine of the algorithm.

#### 4.3 SUPPORT VECTOR MACHINES FORMULATION

In this algorithm the intuition is to start with a random guess of  $w$ , and then iteratively reduce the difference using a *support vector machine* approach. [1]

---

#### **Algorithm 3** *Maximum Margin Separating Hyperplane Algorithm*

---

1. Randomly choose  $w^{(0)}$ . Compute optimal policy  $\pi^{(0)}$  and  $\mu^{(0)} = \mu(\pi^{(0)})$
  2. Compute  $t^{(i)} = \max_{w: \|w\|_2 \leq 1} \min_{j \in \{w \cdot (\mu_E - \mu^{(i)})\}}$  and let  $w^{(i)}$  be the value that attains the maximum.
  3. If  $t^{(i)} \leq \epsilon$ , then terminate.
  4. Using **RL**, compute  $\pi^{(i)}$  wrt  $R^{(i)} = w^{(i)} \cdot \phi$ . *This part is the bottleneck.*
  5. Compute  $\mu^{(i)}$ .
  6.  $i = i + 1$  and repeat.
- 

#### 4.4 MAXIMUM ENTROPY METHODS

Perhaps the most significant development in the IRL paradigm was the introduction of the **Maximum Entropy IRL** algorithm by Brian Ziebart [53]. This method is essentially based on the **Maximum Entropy Principle** [31], which suggests choosing the probability distribution that has maximum information entropy

amongst all distributions that satisfy the data. This is a systematic and tractable way to tackle ambiguity and has seen strong empirical support in physics and machine learning both.

#### 4.4.1 Rationality Model

Rationality Model is a function that defines the probability  $P(b | R)$  that the agent will show behavior (for example, policy), given the reward function  $R$ .

If we know the rationality model of the expert agent, we can use *Bayes Rule* to update our belief about the reward function.

$$P(R | b) = \frac{P(b | R)P(R)}{(\sum_{R'} P(b | R')P(R'))} \quad (4.10)$$

In this sense, the IRL problem for a completely rational expert is equivalent to just maximizing the posterior.

$$\bar{R} = \operatorname{argmax}_R [P(R | b)] \quad (4.11)$$

We know the Max Entropy distribution over trajectories with the constraints of features should have the form

$$P(\tau) = \frac{e^{\mu \cdot \phi(\tau)}}{Z(\mu)}$$

#### 4.4.2 Boltzmann Rationality

An alternative formulation of the sub-optimal behavior of the expert is to assume Boltzmann Rationality - which basically assumes that stupid experts are exponentially less likely.

$$\pi_w(\tau) = \frac{e^{\beta w \cdot \phi(\tau)}}{Z(w)} \quad (4.12)$$

where  $\beta$  is the rationality parameter  $\in [0, \infty)$

For each of the demonstrated trajectories, we update our posterior  $P(w | \tau_i)$  and be left with the beliefs  $\prod_{i=1}^n P(w | \tau_i)$ .

We can then maximize

$$\bar{w} = \operatorname{argmax}_{w \in [0,1]^k} \prod_{i=1}^n P(w | \tau_i) \quad (4.13)$$

The literature notes that sometimes we get better results with a Beta prior (the rationale behind this being that the rewards are incredibly sparse with respect to the state-space in real world), than a uniform prior.

The gradient of the log-likelihood becomes

$$\nabla \mathcal{L}(w) = \mu_D - \sum_{\tau \in \mathcal{D}} P(\tau | w) \phi(\tau) \quad (4.14)$$

We already have the first term from the expected feature count. To get the second term, we use the current estimate of  $w^{(i)}$  to simulate the trajectories using the assumed *rationality model*.

#### 4.4.3 Guided Cost Learning Formulation of Maximum Entropy IRL

In the paper on Guided Cost Learning [18]), Finn et al. provide a slightly more structured approach to the *Maximum Entropy IRL* algorithm. Once again, the assumption here is that experts take actions that are exponentially more likely to generate higher rewards. For a parameterized reward function  $R_\psi(\tau) = \sum_t r_\psi(s_t, a_t)$

$$p(\tau) = \frac{1}{Z} e^{R_\psi(\tau)} \text{ with } Z = \sum_{\tau} e^{R_\psi(\tau)} d\tau \quad (4.15)$$

We want to parameterize the **reward function** in the way that maximizes the log-likelihood

$$L(\psi) = \sum_{\tau \in \mathcal{D}} \log p_r(\psi) \quad (4.16)$$

$$= \sum_{\tau \in \mathcal{D}} \left( \log \frac{1}{Z} + R_\psi(\tau) \right) \quad (4.17)$$

$$= \sum_{\tau \in \mathcal{D}} (R_\psi(\tau)) - M \log Z \quad (4.18)$$

$$= \sum_{\tau \in \mathcal{D}} (R_\psi(\tau)) - M \log \sum_{\tau} e^{R_\psi(\tau)} \quad (4.19)$$

So the gradient wrt the parameters is

$$\begin{aligned} \nabla_{\psi} L(\psi) &= \sum_{\tau \in \mathcal{D}} \frac{dR_{\psi}(\tau)}{d\psi} - M \frac{1}{\sum_{\tau} e^{R_{\psi}(\tau)} d\tau} \sum_{\tau} \tau (e^{R_{\psi}(\tau)} \frac{dR_{\psi}(\tau)}{d\psi}) \\ &= \sum_{\tau \in \mathcal{D}} \frac{dR_{\psi}(\tau)}{d\psi} - M \sum_{\tau} \frac{e^{R_{\psi}(\tau)} dR_{\psi}(\tau)}{\sum_{\tau} e^{R_{\psi}(\tau)}} \end{aligned}$$

$$= \sum_{\tau \in D} \frac{dR_{\psi}(\tau)}{d\psi} - M \sum_{\tau} p(\tau | \psi) \frac{dR_{\psi}(\tau)}{d\psi}$$

Optimizing for  $p(\tau | \psi)$  is same as  $p(s|\psi)$  - otherwise known as **state-visitation frequency**.

Finally

$$L(\psi) = \frac{1}{|D|} \sum_{\tau_d \in D} \frac{dr_{\psi}(\tau_d)}{d\psi} - \sum_s p(s | \psi) \frac{dr_{\psi}(s)}{d\psi}$$

#### 4.4.4 Maximum Causal Entropy Methods

The maximum entropy method described before provides an approximation of the probability distribution of trajectories when the transition dynamics of the environment are stochastic in nature. In any case, if the transition dynamics is irregular enough so that it influences the decisionmaking of the expert, this approximation does not fit anymore. The approach assumes that agents choose actions and not trajectories. In the case of stochastic MDPs, the outcome of any action is not predetermined. For an intuition pump, let us consider the following scenario. Suppose there is one state that has a very high reward but extremely low chances of getting reached. In the previous method, trajectories containing this state would be highly preferred. In the real world, most rational agents would not choose trajectories leading to high reward but low probability of reaching. The maximum entropy model takes into account the reward exponentially and the transition probability only linearly. In experiments as well, [3] MEIRL perform significantly worse when there is high enough stochasticity in the transition dynamics.

To alleviate this issue, the same author proposed an extension to the MEIRL framework by incorporating causal information from the previous steps. This method, called the **Maximum Causal Entropy Inverse Reinforcement Learning (MCEIRL)** [52] maximizes the *causal* entropy.

For a state sequence  $s_{1:T}$  and action sequence  $a_{1:T}$ , causal entropy is defined as

$$\mathcal{H}(a_{1:T} || s_{1:T}) = \sum_{t=1}^T \mathcal{H}(a_t | s_{1:t}, a_{1:t-1}) \quad (4.20)$$

which is the sum of action entropy at all times steps up to the current one, given only the previous states and actions.



With this function to maximize while matching the features leads to the following distribution of actions which uses the  $q$ -values in the exponent.

$$\pi_{\theta}(a | s) \propto e^{[q_{\theta}(s,a)]} \quad (4.21)$$

and the exact probability of action after normalization becomes

$$\pi_{\theta}(a | s) = e^{[q_{\theta}(s,a)-v(s)]} \quad (4.22)$$

The probability of a trajectory finally becomes

$$Pr(\tau | \theta, T) = \prod_{s_t, a_t, s_{t+1} \in \tau} e^{[q_{\theta}(s_t, a_t) - v(s_t)]} \cdot T(s_{t+1} | s_t, a_t) \quad (4.23)$$

This probability expression can be used to maximize the likelihood of expert trajectories as done in the previous section. Crucially, the gradient does not depend on the transition dynamics as the reward parameters  $\theta$  do not influence the dynamics.

## 4.5 BAYESIAN METHODS

Almost all classical [IRL](#) approaches assume the expert to be an optimal rational decision-maker, an assumption that is rarely the case in the real-world [\[30\]](#). Apart from that, as the problem is itself under-specified, they rely on heuristics (albeit principled, for example, in case of [MEIRL](#)) to choose the best estimate of the reward function. These heuristics induce a bias in the estimate nonetheless.

### 4.5.1 Bayesian Inverse Reinforcement Learning

The [Bayesian Inverse Reinforcement Learning \(BIRL\)](#) approach [\[37\]](#) aims to alleviate these two drawbacks by reframing the [IRL](#) problem as a problem of Bayesian inference. This approach aims to calculate the posterior over all possible reward functions with the expert demonstrations as evidence. This also offers a full detailed picture of the reward function as it allows us to infer about the uncertainty about the reward function as well.

Applying Bayes Theorem to model the posterior probability from a prior  $Pr(\hat{R})$  over reward functions

$$Pr(\hat{R} | \mathcal{D}) = \frac{Pr(\mathcal{D} | \hat{R})}{Pr(\mathcal{D})} \quad (4.24)$$

The likelihood  $Pr(\mathcal{D} | \hat{R})$  is based on the model of Boltzman Rationality [45]. We model an imperfect agent choosing actions

$$Pr(a | s, \hat{R}) = \frac{e^{\alpha \cdot \hat{Q}^*(s,a)}}{Z} \quad (4.25)$$

$\alpha$  here is the **rationality** hyperparameter that we can tune with estimates of how perfect the expert is.

The complete likelihood of a data set  $\mathcal{D}$  of trajectories with state-action pairs then is

$$Pr(\mathcal{D} | \hat{R}) = \frac{e^{\alpha \cdot (\sum_{\tau \in \mathcal{D}} \sum_{(s,a) \in \tau} \hat{Q}^*(s,a))}}{Z} \quad (4.26)$$

It is interesting to note that we need not choose a **uniform** prior for the reward function - indeed, there is no strong reason to suspect that the reward functions in real-life RL tasks are **independently identically distributed (i.i.d.)**. The authors instead propose different kinds of priors for the reward function. For example, a **beta distribution** would be suitable if we know that most states have low or negligible rewards, and only a few states have large enough rewards.

Computing the normalization term  $Pr(\mathcal{D})$  in the denominator is not scalable with just several states. The authors use **Markov Chain Monte Carlo (MCMC)** method to approximate the posterior distribution.

#### *Maximum A Posteriori BIRL*

An extension of the **BIRL** framework [12] showed that we could formulate almost all previous **IRL** algorithms in the **BIRL** form by carefully choosing the likelihood function, and doing the inference via a **maximum a posteriori (MAP)** method, rather than using the posterior mean.

This subsumes all previous methods in **IRL** to a broader generality and ability to have a Bayesian inference position.

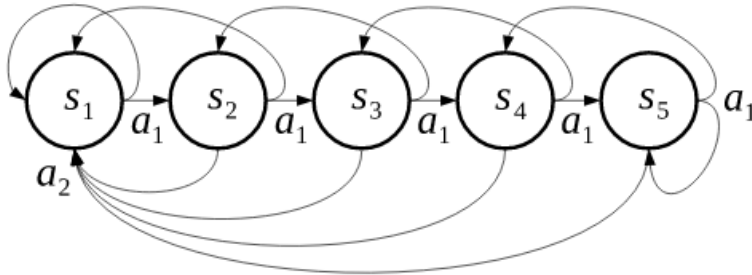


Figure 4.1: A sample five-state MDP. Image courtesy of the MAP-BIRL paper.

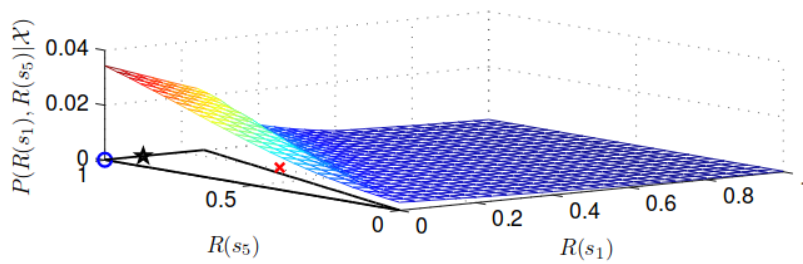


Figure 4.2: The corresponding recovered reward posterior to the five-state MDP as obtained by a MAP-BIRL algorithm. Image courtesy of the MAP-BIRL paper.

Table 4.1: Various IRL algorithms as specific cases of the MAP-BIRL framework.

Algorithm	Likelihood	Prior
Linear Programming IRL	$v^E(R) - v^*(R)$	Uniform
Maximum Margin Planning	$(v^E(R) - v^*(R))^2$	Gaussian
Maximum Entropy IRL	$\log \mathcal{P}_{\maxent}(\mathcal{D} \mid T, R)$	Uniform

In recent years (2010 - 2019), there have been some extraordinary developments in the field of [Inverse Reinforcement Learning \(IRL\)](#), mostly thanks to insights gained from other subfields of machine learning. The astonishing success of [Deep Learning](#) [32] in supervised machine learning tasks around the year 2012 has contributed significantly towards the development of modern [IRL](#) algorithms. Most of these algorithms benefit from the universal capability of deep neural networks to estimate arbitrary functions [23]. With the advantage of hindsight, we can safely suggest that almost all papers in the field since then use deep neural networks in some form or the other.

### 5.1 DEEP LEARNING AND [IRL](#)

The first paper to use deep neural networks to approximately determine the reward function is Maximum Entropy Deep Inverse Reinforcement Learning [51]. The authors used a simplified, fully connected network on top of the framework of Maximum Entropy [IRL](#) to estimate complex and nonlinear reward function.

As DNNs are conducive to gradient methods via backpropagation [22], the reward estimation is effectively similar to the maximum entropy [IRL](#) method. The crucial difference is that that there is no need to assume the linearity of the reward function in the features, as DNNs are generally able to estimate highly nonlinear functions [47].

This approach begins with defining the reward function as a deep neural network (DNN) over the features and unknown weights. The number of hidden layers and the number of neurons in each layer are hyperparameters that can be tuned.

The log-likelihood function, given some demonstrations  $D$ , and model parameters  $\theta$

$$\mathcal{L}(\theta) = \log P(D, \theta | r) = \log P(D|r) + \log P(\theta) \quad (5.1)$$

The first term (dependent on the demonstrations) can be differentiated as

$$\frac{\partial \mathcal{L}_D}{\partial \theta} = \frac{\partial \mathcal{L}_D}{\partial r} \frac{\partial r}{\partial \theta} \quad (5.2)$$

Now, from the maximum entropy framework, we already know that

$$\frac{\partial \mathcal{L}_D}{\partial r} = \mu_D - \mathbb{E}[\mu] \quad (5.3)$$

Then, we can construct the whole network loss and update the reward function estimate using backpropagation.

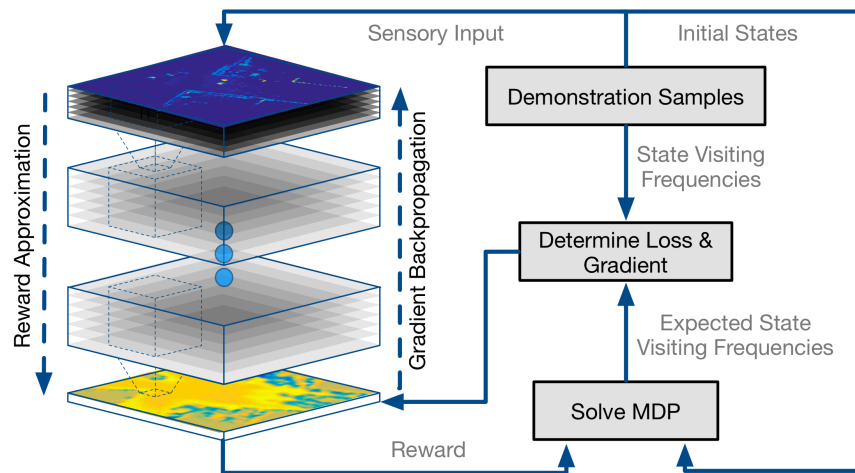


Figure 5.1: The Deep MEIRL pipeline. Image obtained from the original paper.

## 5.2 GANS AND IRL

A **Generative Adversarial Network (GAN)** is an adversarial (two subsections of the unit compete with each other) approach towards solving various generative (in opposition with discriminative) machine learning tasks. A GAN consists of two separate artificial neural networks (discriminator and generator) that are trained together. In the original formulation of the GAN, given a true data distribution, the generator learns to generate real-looking samples, as an adversary to the discriminator that tries to discern whether a sample is true or generated.<sup>5</sup>

Two almost similar approaches (cite GAIL and GAN-GCL) have formulated IRL as a GAN-like problem.

<sup>5</sup> Though the primary success of GAN have been in image generation (for example, <https://www.thispersondoesnotexist.com/>), the framework itself is general and domain agnostic.

In the maximum entropy IRL framework, estimating the partition function does not scale with the complexity of the environment. **Guided Cost Learning (GCL)** [18] uses importance sampling to approximate this partition function.

### 5.3 ADVERSARIAL METHODS

#### 5.3.1 Empowerment Based Adversarial Inverse Reinforcement Learning

As of April 2019, perhaps the state-of-the-art IRL method is the EAIRL algorithm [36].

This method learns simultaneously nearly optimal rewards and policy, in contrast to the previous **Generative Adversarial Imitation Learning (GAIL)** method. Being an imitation learning approach GAIL recovers policy only, not transferrable rewards. In this approach, the authors make use of **Empowerment** - an information-theoretic measure somewhat similar to the KL divergence. Empowerment is a **mutual information** based potential function, like value functions, which intuitively quantifies for a state the extent to which an agent can influence its environment. Empowerment acts as a regularizer in policy updates, apart from resolving the reward function.

Empowerment is a maximal of mutual information between a sequence of  $K$  actions  $a$  and the final state  $s'$  reached after the execution of those actions, conditioned on current state  $s$ .

$$\Phi(s) = \max I(a, s' | s) = \max \mathbb{E}_{p(s'|a,s)w(a|s)} \left[ \log \left( \frac{p(a, s' | s)}{w(a|s)p(s'|s)} \right) \right]$$

The authors approximate Empowerment and finally optimize it using the loss function

$$l_I(s, a, s') = |\beta \log q_\phi(a|s', s) - (\log \pi_\theta(a|s) + \Phi_\phi(s))|$$

The algorithm has **four** models that are trained simultaneously. This model is inherently dependent on the **Generative Adversarial Network (GAN)** framework, as a derivative of GAIL.

1. **inverse model** (maximum log-likelihood supervised learning problem) that, given a set of trajectories, minimizes the mean-square error between its predicted action  $q(a|s', s)$  and the action  $a$  according to the generated trajectory.

$$l_q(s, a, s') = (q_\phi(\cdot | s, s') - a)^2$$

2. **empowerment optimization** as noted before
3. **reward function** first compute the Discriminator as

$$D_{\zeta,\varphi}(s, a, s') = \frac{\exp[r_{\zeta}(s, a) + \gamma\Phi_{\varphi'}(s') - \Phi_{\varphi}(s)]}{\exp[r_{\zeta}(s, a) + \gamma\Phi_{\varphi'}(s') - \Phi_{\varphi}(s)] + \pi_{\theta}(a|s)}$$

Finally train the parameters  $\zeta$  by binary logistic regression to discriminate between expert and generated trajectories via

$$\min_G \max_D \mathbb{E}_{\tau}[\log D_{\zeta,\varphi}(s, a, s')] + \mathbb{E}_{\tau_E}[(1 - \log D_{\zeta,\varphi}(s, a, s'))]$$

4. **policy optimization**

train the policy  $\pi_{\theta}(a|s)$  to maximize the discriminative reward  $\hat{r}(s, a, s') = \log D(s, a, s') - \log(1 - D(s, a, s'))$  and to minimize the loss function  $l_I(s, a, s') = |\beta \log q_{\varphi}(a|s, s') - (\log \pi_{\theta}(a|s) + \Phi_{\varphi}(s))|$  which accounts for empowerment regularization overall training objective becomes

$$\mathbb{E}_{\pi}[\log \pi_{\theta}(a|s)\hat{r}(s, a, s')] + \lambda_I \mathbb{E}_{\tau}[l_I(s, a, s')]$$

---

#### Algorithm 4 EAIRL

---

1. Initialize parameters of policy  $\pi_{\theta}$ , and inverse model  $q_{\psi}$
  2. Initialize parameters of target  $\phi_{\psi'}$  and training  $\phi_{\psi'}$  empowerment, and reward  $r_{\zeta}$  functions
  3. Obtain expert demonstrations  $\tau_E$
  4. for  $i$  in range(N)
    - a) Collect trajectories
    - b) Update all four models with respective gradients.
    - c) After  $n$  epoch sync with target.
-



Table 5.1: Comparison of various IRL algorithms in terms of key properties

Method	Expert	Reward Function	Heuristics	Solving MDP	Transition Dynamics
Linear Programming	Optimal	Linear	Distance	IRL Iteration	Known
Maximum Margin	Optimal	Linear	Feature Matching	IRL Iteration	Known
Bayesian	Softmax	Tabular	Posterior Mean	MCMC	Unknown
MAP Bayseian	Flexible	Flexible	Posterior Mode	IRL Iteration	Flexible
Maximum Entropy	Softmax	Linear	Entropy	IRL Iteration	Known
Maximum Causal Entropy	Softmax	Linear	Causal Entropy	IRL Iteration	Known
Guided Cost Learning	Softmax	Nonlinear	MEIRL + GANs	Once	Unknown
Empowerment IRL	Softmax	Nonlinear	MCEIRL + GANs	Once	Unknown



## TOWARDS NEW IRL ALGORITHMS

## 6.1 INVERSE REINFORCEMENT LEARNING WITH RANKED TRAJECTORIES

We have already discussed in 3.5.1 the underspecified nature of the IRL problem. Though there are systematic ways to deal with this, such as 4.4, all of those are heuristics and exploit regularity in the common IRL tasks, and do not offer any domain-agnostic theoretical guarantee.

How could we try to go about solving this issue? From a mathematical standpoint, the classic formulation of the IRL problem is bound to have this problem, pretty much by definition itself. Can we reformulate the problem to reduce the underdefinedness somehow? It seems that we must include more information in the problem statement in order to have more precise solutions.

The standard IRL formulation uses a set of *expert* demonstrations to infer about the reward function. Even if, in reality, the demonstrations are not optimal, almost all algorithms assume that these are indeed near-optimal <sup>7</sup>.

In any real-world scenario, we are likely to obtain expert demonstrations that vary in *quality* - trajectories that range over the various degree of capability. Even if we only have optimal demonstrations, we can add noise to it to generate suboptimal ones. Can we provide an IRL algorithm some idea about how *good* or *bad* a trajectory is? One way to go about that would be to rank the trajectories. An approximation to that would *binning* the trajectories into several classes - like star ratings of one star to five stars.

The core motivation for the approach is, *'how can we extract the maximum amount of information about the reward function if we are given a ranked set of 'good' and 'bad' trajectories?'*

## 6.1.1 Related Work

Using suboptimal demonstrations for IRL is a relatively recent direction in the field. Previously there have been attempts at using *failed demonstrations* in the LfD community. However, this

<sup>7</sup> One interesting exception is 4.5.1, which uses a rationality parameter to make room for sub-optimal experts but there is only one class of trajectories with same implied rationality.

approach required explicit labelling of failed attempts and the algorithm learns from two clusters of failed and successful attempts [21]. Another method tries to identify anomalous demonstrations from a small number of suboptimal demonstrations [13].

### *Trajectory-Ranked Reward Extrapolation*

Using suboptimal demonstrations and ranking simultaneously to extrapolate to better-than-human performance has been first shown in the T-REX algorithm. [9]. The goal of the algorithm is to find a parametrized reward function  $\hat{r}_\theta$  which explains the ordering of the trajectory, and potentially extrapolates to near-optimal ground truth reward function, even if the trajectories are far from optimal. This algorithm compares two trajectories along with their ranks to learn a reward function that explains the ranking, considering a comparison between the total return of two trajectories under various reward function estimates. It represents the reward of a state as a neural network and constructs a trainable loss function as follows.

1. approximate  $\hat{r}_\theta$  by a neural network such that

$$\sum_{s \in \tau_i} \hat{r}_\theta(s) < \sum_{s \in \tau_j} \hat{r}_\theta(s) \text{ whenever } \tau_i \prec \tau_j$$

2. use the generalized loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau_i, \tau_j \sim \mathcal{D}} [\zeta(\mathbb{P}(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j), \tau_i \prec \tau_j))] \quad (6.1)$$

where  $\zeta$  is a binary classification loss function (the paper uses cross-entropy) and

$$\hat{J}_\theta(\tau) = \sum_{s \in \tau} \gamma^t r(s) \quad (6.2)$$

3. represent the probability  $\mathbb{P}$  as a softmax-normalized distribution

$$\mathbb{P}(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j)) = \frac{e^{\sum_{s \in \tau_j} \hat{r}_\theta(s)}}{e^{\sum_{s \in \tau_j} \hat{r}_\theta(s)} + e^{\sum_{s \in \tau_i} \hat{r}_\theta(s)}} \quad (6.3)$$

4. the loss function becomes

$$\mathcal{L}(\theta) = - \sum_{\tau_i \prec \tau_j} \log \frac{e^{\sum_{s \in \tau_j} \hat{r}_\theta(s)}}{e^{\sum_{s \in \tau_j} \hat{r}_\theta(s)} + e^{\sum_{s \in \tau_i} \hat{r}_\theta(s)}} \quad (6.4)$$

This loss function trains a classifier that can predict whether one trajectory is preferable to another based on the predicted returns of each trajectory.

Given the learned reward function  $\hat{r}_\theta(s)$ , T-REX then seeks to optimize a policy with better-than-demonstrator performance through reinforcement learning using the learned reward function, thus performing imitation learning as well.

T-REX differs from our approach in several aspects. Firstly, T-REX uses comparison between two trajectories as ranks, rather than a full ranking of all  $m$  trajectories into  $1, \dots, m$ . It is essentially training a classifier to distinguish between two trajectories by proxying for the total return of the trajectories under some reward function. Secondly, it uses a neural network to estimate the reward function and tunes it to minimize the loss function, while our approach is more general where potentially any representation of the reward function can be trained.

### *Inverse Reinforcement Learning with Multiple Ranked Experts*

A more general approach than T-REX is the framework of *Inverse Reinforcement Learning with Multiple Ranked Experts* [10] which is also similar to our approach. It obtains demonstrations from a set of ranked experts of several ranked experts of variable optimality. Using ideas from ordinal regression, it strives to obtain to maximize the *margin* between to trajectories of different ranks.

Our approach is slightly different in the sense that we don't essentially maximize the margin between two trajectories. Rather, we try to match the distribution of states in different qualities of demonstrations. Our goal is similar to the idea of feature matching described in section 4.2.

#### 6.1.2 *Definitions*

We define a couple of new functions required for the algorithms.

Like the original formulation of the problem, we assume we are given a set of  $m$  **unranked** trajectories  $\mathcal{D}$ .

We use a metric  $d$  that calculates ‘distance’ between two trajectories (with same starting state), defined as

$$d(\tau_i, \tau_j) = \begin{cases} 0 & \tau_i = \tau_j \\ \max\{\text{len}(\tau_i), \text{len}(\tau_j)\} - 2 & s_1^i = s_1^j, s_T^i = s_T^j, s^i \neq s^j \text{ otherwise} \\ \max\{\text{len}(\tau_i), \text{len}(\tau_j)\} - 1 & s_1^i = s_1^j, s^i \neq s^j \text{ otherwise} \\ d(\tau_{i1}, \tau_{j1}) + d(\tau_{i2}, \tau_{j2}) + \dots + d(\tau_{iv}, \tau_{jv}) & \text{piecing } \tau_1 \text{ and } \tau_2 \text{ accordingly} \end{cases} \quad (6)$$

This metric calculates how different two trajectories are in terms of the states belonging to them, even when they are different in length, and even when they have different final state. This metric lets us assign a low value when most of the states are same and a very high value when most of states are different.

The choice of this metric is just a preliminary one. There might be better options such as the existing and well-known distance measure *edit-distance* [38].

#### Definition 1: Auxiliary Functions

$\rho_H(\tau)$  is a ranking function that assigns a rank between 1 and  $m$  to each trajectory. (this can be considered as a human ranking good vs. bad trajectories.)

A simple scaling function  $l_H(\tau) = 1 - \frac{\rho_H(\tau)-1}{m}$ .

$G(\tau) = \sum_{s \in \tau} r(s)$  is the total return of a trajectory. (Can be modified to include discount factor.)

$\rho_A(\tau)$  is another function that assigns ranking when Reward function is via ordering the trajectories by total return  $G(\tau)$ .

Similarly another function  $l_A(\tau) = 1 - \frac{\rho_A(\tau)-1}{m}$ .

#### Definition 2: Rank Weighted State Distribution (RWSD)

For a particular choice of dataset  $D$  and ranking function  $l$ , we define the **rank-weighted state distribution** as

$$\phi_l^D(s) = \frac{1}{m} \sum_{s \in \tau} l(\tau) \quad \forall s \in \mathcal{S}$$

### 6.1.3 Algorithm 1: Rank Difference as Reward Function Proxy

This is a fairly simple algorithm where we are motivated by the fact that if two very similar trajectories have a large rank gap, the states that are different between those two must be responsible for that

For all states  $s \in \mathcal{S}$ ,

$$r(s) = \sum_{\substack{\tau_i, \tau_j \\ s \in \tau_i \\ s \notin \tau_j}} \frac{l_H(\tau_i) - l_H(\tau_j)}{d(\tau_i, \tau_j)}$$

This is essentially using the rank difference (scaled to comparable domains using the scaling function) as a proxy for the reward function. For any state, we consider all trajectory pairs (complexity of order  $O(m^2)$  where one contains the state and the other does not. We calculate the difference between their ranks and divide by the distance between the trajectories. As a result of this, if two trajectories differ hugely in rank and just few state, then those states get reward proportional to their rank difference, scaled by the amount of difference between the trajectories.

### 6.1.4 Algorithm 2: Bayesian RWSD Matching

This algorithm is Bayesian in nature and can provide richer information about the reward function. This also has the machinery to deal with the possibility of a sub-optimal human ranker, using the concept of Boltzman rationality.

The essential idea is similar to that of feature matching as described in section 4.2. We strive to train the agent in a manner that it matches the *RWSD* between the original demonstrations and the *RWSD* calculated using its reward estimate.

As the definition of  $\phi_l^D$  depends on the choice of the ranking function  $l$ , our goal is to learn a reward function that makes  $\phi$  similar for the human ranking  $l_H$  (or the empirical one) and  $l_A$  the ranking calculated from the current reward function estimate.

#### *Likelihood*

We model the expert's (human) *ranking capability* as a **Boltzman Rationality** [41] distribution with rationality parameter  $\beta$ .

We model the likelihood of a particular rank weighted state distribution, given a reward function estimate  $R$  is

$$P(\phi | R) \propto e^{-\frac{\eta}{\beta}} \quad (6.6)$$

where  $\eta = JS(\phi_{I_H} || \phi_{I_A})$  is the Jensen-Shannon divergence [33] between these two distributions.

*Posterior*

Apply Bayes Rule to get the Posterior

$$P(R | \phi) = \frac{1}{Z} P(\phi | R) P(R) \quad (6.7)$$

It is to be noted that finding the normalization constant can be challenging. However, as it depends on the reward function distribution only, it is as challenging as the previous formulation of Bayesian IRL which solves the RL problem in the inner loop of each MCMC step in the IRL problem.

### 6.1.5 Experiments

*Setup*

We experimented on the first algorithm using the FrozenLake-v0<sup>8</sup> reinforcement learning environment from OpenAI Gym [8], using 20000 trajectories generated by a value iteration RL algorithm, ranked by trajectory length and total reward gained.

The FrozenLake environment is a basic toy-text reinforcement learning setup. It comes in two variants of 4x4 and 8x8 grids of 16 and 64 states. We used the first variant for our experiments.

The agent controls the movement in the gridworld described above. There are a few ‘holes’ that ends the episode and the rest of the cells are walkable. The movement direction of the agent is uncertain and only partially depends on the chosen direction. The probability of going to a particular direction is determined by the transition dynamics matrix. The agent receives a reward only for reaching the goal state.

The 4x4 grid can be described like the following.

<sup>8</sup> The original documentation builds a story around this environment as following. There is a frisbee located somewhere in a frozen lake. There is significant amount of wind and the surface is slippery.



S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

(S: starting point, safe)

(F: frozen surface, safe)

(H: hole, fall to your doom)

(G: goal, where the frisbee is located)

### Results

We ran the algorithm 40 times with 20000 trajectories every run. In each run, we ranked the expert trajectories (obtained by a value iteration RL algorithm) by total reward collected and length of trajectory (thus incentivizing for learning to reach the goal as soon as possible). We normalized the learned reward function to  $[0, 1]$ . Below we report the mean reward function and the standard deviation.

We can clearly assert that the very low standard deviation indicates that the algorithm converges well.

If we compare the reward value for each state, it is evident that the algorithm correctly identifies the ‘holes’ as of being zero reward. The ‘frozen’ cells are of non-zero reward indicating that a forward learner must walk through them. The first starting state is assigned a reward of 1 since it belongs to every trajectory.

The first algorithm obtained the following reward estimate.

```
mean_reward =
[[1.    , 0.02, 0.05, 0.04],
 [0.75, 0.   , 0.02, 0.   ],
 [0.54, 0.31, 0.11, 0.   ],
 [0.   , 0.28, 0.24, 0.09]]

reward_std =
[[0, 5.87e-04, 1.28e-03, 1.46e-03],
 [3.07e-03, 9.10e-05, 4.37e-04, 0],
 [3.60e-03, 2.53e-03, 1.09e-03, 0],
 [0, 2.74e-03, 2.51e-03, 8.53e-04]]
```

Though this reward function looks different from the original one, it would logically induce the same behavior (policy) and incur low inverse learning error.

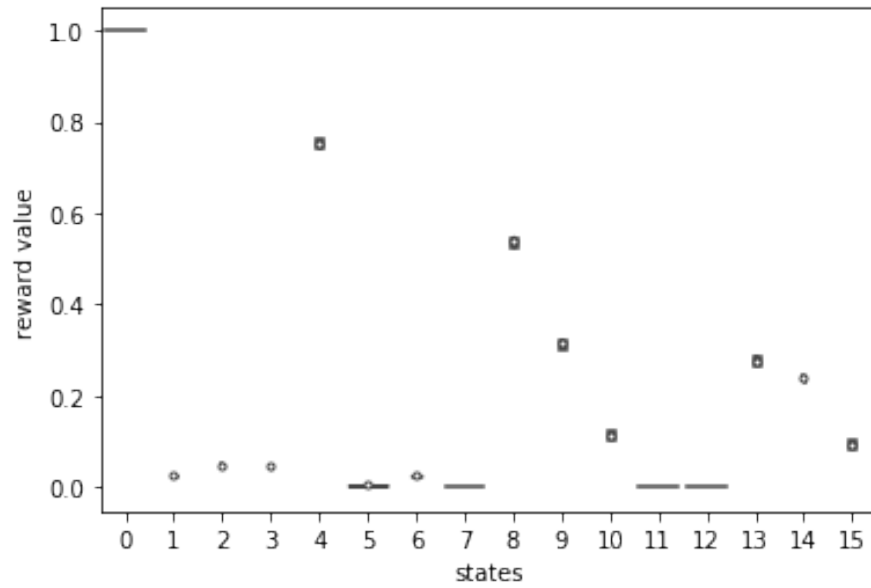


Figure 6.1: Violinplot for the reward function obtained in the 40 runs of the algorithm.

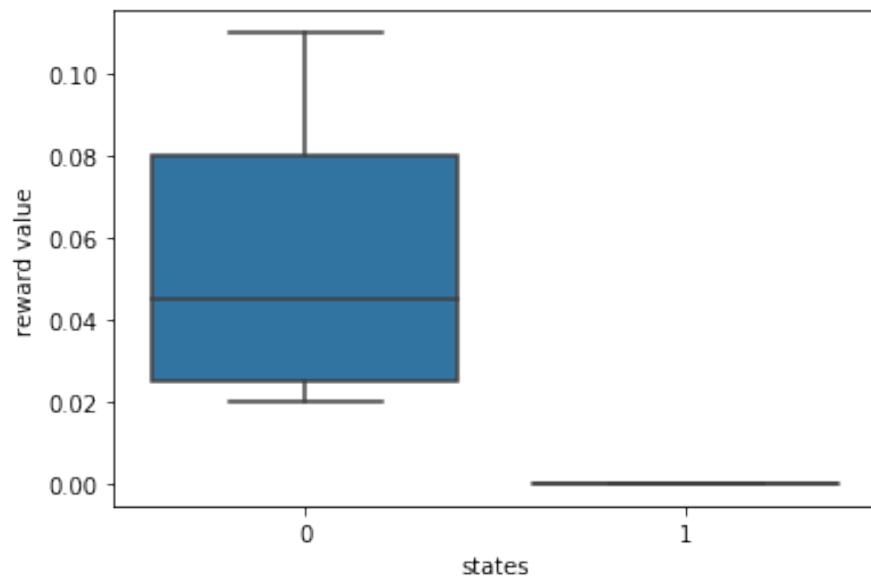


Figure 6.2: Boxplot of *only* the low reward values of the frozen states vs all the zero reward values of the hole cells. Frozen states correspond to 0 and holes correspond to 1.

Figures 6.1 and 6.2 illustrates that though the reward values of the frozen states are positive albeit highly distributed (which does not matter due to the reward satisfiability condition - a forward RL algorithm would make use of the this reward values to find the shortest path to the goal, irrespective of how small an

individual reward value is, as long as it is more than zero), the hole states are all assigned clearly distinguishable being zero without any variance.

Obviously, this method can fail and not scale in more complex environments as the size of the state-space grows and transition dynamics affects the behavior heavily.

#### 6.1.6 *Comments*

The scaling function, in contrast to the pairwise comparisons such as T-REX, provides a ground for comparison of all the trajectories in a normalized setting. We essentially consider all the trajectory ranks, irrespective of the total number of trajectories, into the segment  $[0, 1]$ . This scaling function might fail significantly if we do not have a set of demonstrations that are uniformly drawn from a continuously varying expert optimality.

## 6.2 CONDITIONAL GANS AND RANKED IRL

As discussed in the previous chapter, the recent advances in [Generative Adversarial Network \(GAN\)](#)s have opened a new avenue for using generative algorithms in problems of under-specification leading to an adversarial formulation that alleviates many of the traditional challenges of the field, of course also thanks to modern GPU compute capabilities.

There is a natural way to augment ranked or ‘binned’ trajectories into the [GAN](#) framework as detailed below. Currently, this is an untested new idea only, and not a full-fledged algorithm, either theoretically or experimentally.

Conditional [GANs](#) are an extension to the GAN framework where the data is labeled into different classes. The discriminator tries to identify both the realness and label of a sample, and the generator tries to generate data in specific classes.

We propose to setup a conditional [GAN](#) with an architecture that can parameterize the trajectories with the reward function as a deep neural network. The generators goal is to generate trajectories that look very much like the expert trajectories, labeled with the class or rank of the expert. The discriminator’s job is to discern whether a labeled sample is true or generated. We can construct a loss based on the different ranks or classes. Essentially we will be able to backpropagate to the reward parameters

itself and thus the reward function. This method will again allow extrapolation of sub-optimal demonstrations beyond the expert's capability.

To implement an algorithm like this, one has to go through the following steps. Being a nascent idea only, there are crucial steps that need to be figured out, as mentioned below.

1. Input ground truth data of  $m$  classes of expert demonstration trajectories. Expert optimality for each class is consistent but there is an order among the classes themselves from worst to best.

Current conditional GANs treat data classes without any ordinal relationships, and we need to figure out how to integrate this information meaningfully.

2. The generator is tasked to generate labeled trajectories which resemble the ground truth data. Specifically it outputs a trajectory and the class it belongs to, as realistically as possible. So, it not only needs to emulate real-looking trajectories, it needs to do that in-class for all classes.

The generator is a neural network that has its parameters which is also the parameter for the reward function. *We need to find the best way to parametrize the reward function with the same weights for the neural network.* One way to do that might be to construct a reward function from the neural network weights themselves, and then use that reward function to train a forward RL algorithm to generate trajectories for each optimality class. While this is a simple approach, this does not avoid the problem of solving the forward RL problem in the inner loop, and there might be a better clever solution to this.

3. The discriminator tries to discern between real and generated data just like the original GAN problem and returns a discriminator loss.
4. The gradient of the loss for both the networks are differentiable just like the original setup. Update rules and other hyperparameters also are alike.

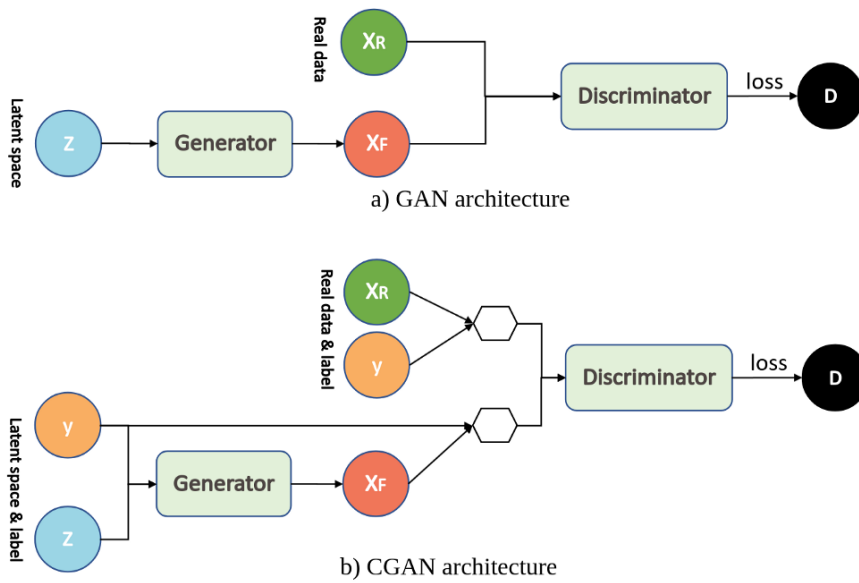


Figure 6.3: A comparison of the architectures of GANs and Conditional GANs. Image courtesy: Lilian Weng.



## CONCLUSIONS

---

In this chapter, the materials of this thesis are reviewed and put into the context of previous academic work in the field. This is followed by a discussion of potential future extensions to what has been achieved.

### 7.1 DISCUSSIONS

The primary objective of this thesis to evaluate and extend the field of [Inverse Reinforcement Learning \(IRL\)](#) has been dealt with in detail in the last six chapters. Here we discuss a birds-eye-view of the breakthroughs of the field as a whole and the yet-to-solve shortcomings.

#### 7.1.1 *Breakthroughs*

Through the use of principles-based techniques, such as the optimization of maximum entropy, maximum margin, and the regression in the Gaussian process, solution methods have made significant progress in addressing the primary challenge of being a problem of poorly restricted learning. The machine learning problem due to its large hypothesis space has been significantly mitigated by choosing a reward function linear in features. Although this imposed structure limits the class of hypotheses, it often finely expresses the reward function in many real-world domains. Importantly, it allowed the use of feature expectations as sufficient statistics to represent the value of the trajectories or the value of an expert's policy. This has contributed significantly to the success of the first methods, such as the SVM-IRL. This is also observed in approaches based on maximum probability, such as [MEIRL](#). Maxent distributes the probability mass based on entropy but under the restriction of the matching of feature expectations.

### 7.1.2 Shortcomings

During my yearlong journey of studying the domain of [IRL](#), several distinct but essential shortcomings of the field have surfaced. These are not shortcomings of the *algorithm* or theory of the field, but rather shortcomings of how the field is practiced. I want to expound upon a few of these operational challenges in the next couple of paragraphs.

Despite being a field that has been around for about twenty years, there is no standard benchmark set of problems for [IRL](#). There is no standard framework for implementing [IRL](#) algorithms or an environment. For the forward problem of [Reinforcement Learning \(RL\)](#), there exists many frameworks [20, 11, 50, 40, 15] and environments [8] making it one of the most happening sub-field of machine learning.<sup>10</sup>

Also, so far, very few efforts have been made to analyze the time-complexity or other standard machine learning efficiency comparisons in this domain. While very incremental progress in any of these in the case of deep [Reinforcement Learning \(RL\)](#) makes regular headlines, the scene for [IRL](#) research is parsimonious.

Finally, although the vision of the co-founder of the field, Stuart Russell is to apply [IRL](#) to reformulate the whole domain of machine learning to a framing that is conducive to solving the AI Alignment problems<sup>11</sup>, this effort is limited to a small circle of researchers. Hopefully, Russell's very recently published book on the subject — *Human Compatible - Artificial Intelligence and the Problem of Control* [39] can focus more light on this and bring a diverse and extensive effort towards solving this problem.

## 7.2 FUTURE WORK

Firstly, we intend to finish the nascent ideas introduced in chapter 6 and investigate more about the properties and implications of such a formulation.

Secondly, and more importantly, we need a new language to *talk* about problems where underspecification is the crux. Currently, we have no way to classify and quantify underspecification more than the very basics. It seems apt that an expressive language can help researchers formulate the problems and reach the epitome much more effectively. Methods of applied category theory, such as generative effects [19] may be potent here, but

<sup>10</sup> One partial solution is the Python package `irl-benchmark` co-developed by the current author of this thesis. For more information, see <https://github.com/JohannesHeidecke/irl-benchmark>.

<sup>11</sup> more precisely the problem of *value learning* but it is not unimaginable that it can be extended to other problems such as *wireheading* [34]



along with that, better experimental abstractions of functional programming might also come handy.

Thirdly, can we find a bound or condition on number or type of the expert trajectories relating to  $\leq \epsilon$  *inverse learning error*?

Fourthly, it would be useful and interesting to create a unified framework for all types of IRL algorithms incorporating ranked trajectories.

Lastly, can we create a human-in-the-loop algorithm which queries an expert to rank a few trajectories at each step? This will readily reduce the number of trajectories an expert has to actually perform and make the whole procedure more efficient. If we can come up with some 'confidence' parameter for the reward of each state (or the reward parameters), the algorithm can query the human to rank trajectories which have a disparity between the states it is least confident about. Using that information, the algorithm can use methods of divergence with the samples to efficiently use the ranking information.



## APPENDIX



This is a sample implementation of the 4.4 algorithm for reference purpose. This algorithm is part of the Python IRL benchmarking suite `irl-benchmark` co-implemented by the current author. More details can be found at <https://github.com/JohannesHeidecke/irl-benchmark>.

#### A PYTHON MEIRL IMPLEMENTAION

```

"""Module for maximum entropy inverse reinforcement learning."""

from typing import Callable, Dict, List

import gym
import numpy as np

from irl_benchmark.config import IRL_CONFIG_DOMAINS, IRL_ALG_REQUIREMENTS
from irl_benchmark.irl.algorithms.base_algorithm import BaseIRLAlgorithm
from irl_benchmark.irl.feature.feature_wrapper import FeatureWrapper
from irl_benchmark.irl.reward.reward_wrapper import RewardWrapper
from irl_benchmark.metrics.base_metric import BaseMetric
from irl_benchmark.rl.algorithms.base_algorithm import BaseRLAlgorithm
from irl_benchmark.rl.model.model_wrapper import BaseWorldModelWrapper
from irl_benchmark.utils.wrapper import unwrap_env

class MaxEntIRL(BaseIRLAlgorithm):
    """Maximum Entropy IRL (Ziebart et al., 2008).

    Not to be confused with Maximum Entropy Deep IRL (Wulfmeier et al., 2016)
    or Maximum Causal Entropy IRL (Ziebart et al., 2010).
    """

    def __init__(self, env: gym.Env, expert_trajs: List[Dict[str, list]],
                 rl_alg_factory: Callable[[gym.Env], BaseRLAlgorithm],
                 metrics: List[BaseMetric], config: dict):
        """See :class:`irl_benchmark.irl.algorithms.base_algorithm.BaseIRLAlgorithm`."""

        super(MaxEntIRL, self).__init__(env, expert_trajs, rl_alg_factory,
                                         metrics, config)

        # get transition matrix (with absorbing state)
        self.transition_matrix = unwrap_env(
            env, BaseWorldModelWrapper).get_transition_array()
        self.n_states, self.n_actions, _ = self.transition_matrix.shape

        # get map of features for all states:
        feature_wrapper = unwrap_env(env, FeatureWrapper)
        self.feats_map = feature_wrapper.feature_array()

    def expected_svf(self, policy: np.ndarray) -> np.ndarray:
        """Calculate the expected state visitation frequency for the trajectories
        under the given policy. Returns vector of state visitation frequencies.
    """

```

Uses `self.transition_matrix`.

#### Parameters

-----

`policy`: `np.ndarray`

The policy for which to calculate the expected SVF.

#### Returns

-----

`np.ndarray`

Expected state visitation frequencies as a numpy array of shape `(n_states,`

`"""`

```
# get the length of longest trajectory:
```

```
longest_traj_len = 1 # init
```

```
for traj in self.expert_trajs:
```

```
    longest_traj_len = max(longest_traj_len, len(traj['states']))
```

```
# svf[state, time] is the frequency of visiting a state at some point of time
```

```
svf = np.zeros((self.n_states, longest_traj_len))
```

```
for traj in self.expert_trajs:
```

```
    svf[traj['states'][0], 0] += 1
```

```
svf[:, 0] = svf[:, 0] / len(self.expert_trajs)
```

```
for time in range(1, longest_traj_len):
```

```
    for state in range(self.n_states):
```

```
        total = 0
```

```
        for previous_state in range(self.n_states):
```

```
            for action in range(self.n_actions):
```

```
                total += svf[
```

```
                    previous_state, time - 1] * self.transition_matrix[
```

```
                        previous_state, action, state] * policy[
```

```
                            previous_state, action]
```

```
                svf[state, time] = total
```

```
# sum over all time steps and return SVF for each state:
```

```
return np.sum(svf, axis=1)
```

```
def train(self, no_irl_iterations: int,
```

```
           no_rl_episodes_per_irl_iteration: int,
```

```
           no_irl_episodes_per_irl_iteration: int):
```

```
    """Train algorithm. See abstract base class for parameter types."""
```

```
# calculate feature expectations
```

```
expert_feature_count = self.feature_count(self.expert_trajs, gamma=1.0)
```

```
# start with an agent
```

```
agent = self.rl_alg_factory(self.env)
```

```
reward_wrapper = unwrap_env(self.env, RewardWrapper)
```

```
theta = reward_wrapper.reward_function.parameters
```

```

irl_iteration_counter = 0
while irl_iteration_counter < no_irl_iterations:
    irl_iteration_counter += 1

    if self.config['verbose']:
        print('IRL ITERATION ' + str(irl_iteration_counter))
    # compute policy
    agent.train(no_rl_episodes_per_irl_iteration)

    policy = agent.policy_array()

    # compute state visitation frequencies, discard absorbing state
    svf = self.expected_svf(policy)[: -1]

    # compute gradients
    grad = (expert_feature_count - self.feats.T.dot(svf))

    # update params
    theta += self.config['lr'] * grad

    reward_wrapper.update_reward_parameters(theta)

    evaluation_input = {
        'irl_agent': agent,
        'irl_reward': reward_wrapper.reward_function
    }
    self.evaluate_metrics(evaluation_input)

return theta

```

```

IRL_CONFIG_DOMAINS[MaxEntIRL] = {
    'verbose': {
        'type': bool,
        'default': True
    },
    'lr': {
        'type': float,
        'default': 0.02,
        'min': 0.000001,
        'max': 50
    }
}

```

```

IRL_ALG_REQUIREMENTS[MaxEntIRL] = {
    'requires_features': True,
    'requires_transitions': True,
}

```





## ACRONYMS

---

- BIRL** Bayesian Inverse Reinforcement Learning. 41, 42
- DL** Deep Learning. 18, 45
- GAIL** Generative Adversarial Imitation Learning. 47
- GAN** Generative Adversarial Network. 18, 46, 47, 59, 60
- GCL** Guided Cost Learning. 47
- i.i.d.** independently identically distributed. 19, 42
- IRL** Inverse Reinforcement Learning. 6, 9, 17, 18, 27–29, 33, 41, 42, 45–47, 51, 56, 63–65, 69
- LfD** Learning from Demonstrations. 51
- MAP** maximum a posteriori. 42
- MCEIRL** Maximum Causal Entropy Inverse Reinforcement Learning. 40
- MCMC** Markov Chain Monte Carlo. 42
- MDP** Markov Decision Process. 21, 24, 40
- MEIRL** Maximum Entropy Inverse Reinforcement Learning. 40, 41, 63
- RL** Reinforcement Learning. 18–20, 26, 37, 42, 56, 64



## BIBLIOGRAPHY

---

- [1] Pieter Abbeel and Andrew Y. Ng. “Apprenticeship learning via inverse reinforcement learning.” In: *Twenty-first international conference on Machine learning - ICML '04* (2004). DOI: [10.1145/1015330.1015430](https://doi.org/10.1145/1015330.1015430). URL: <http://dx.doi.org/10.1145/1015330.1015430>.
- [2] Pieter Abbeel and Andrew Y. Ng. “Inverse Reinforcement Learning.” In: *Encyclopedia of Machine Learning and Data Mining* (2016), pp. 1–5. DOI: [10.1007/978-1-4899-7502-7\\_142-1](https://doi.org/10.1007/978-1-4899-7502-7_142-1). URL: [http://dx.doi.org/10.1007/978-1-4899-7502-7\\_142-1](http://dx.doi.org/10.1007/978-1-4899-7502-7_142-1).
- [3] Adria Garriga Alonso, Max Marian Daniel, Johannes Heidecke, Anton Osika, Sayan Sarkar. *IRL Benchmark - A Python Package for Benchmarking Inverse Reinforcement Learning Algorithms*. URL: <https://github.com/JohannesHeidecke/irl-benchmark>.
- [4] Kareem Amin and Satinder Singh. *Towards Resolving Unidentifiability in Inverse Reinforcement Learning*. 2016. arXiv: [1601.06569v1](https://arxiv.org/abs/1601.06569v1) [cs.AI].
- [5] Brenna D Argall et al. “A survey of robot learning from demonstration.” In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.
- [6] Stuart Armstrong and Sören Mindermann. *Occam’s razor is insufficient to infer the preferences of irrational agents*. 2017. arXiv: [1712.05812v6](https://arxiv.org/abs/1712.05812v6) [cs.AI].
- [7] Richard Bellman et al. “The theory of dynamic programming.” In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515.
- [8] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: [arXiv: 1606.01540](https://arxiv.org/abs/1606.01540).
- [9] Daniel S Brown et al. “Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations.” In: *arXiv preprint arXiv:1904.06387* (2019).
- [10] Pablo Samuel Castro, Shijian Li, and Daqing Zhang. *Inverse Reinforcement Learning with Multiple Ranked Experts*. 2019. arXiv: [1907.13411](https://arxiv.org/abs/1907.13411) [cs.LG].

- [11] Pablo Samuel Castro et al. “Dopamine: A Research Framework for Deep Reinforcement Learning.” In: (2018). URL: <http://arxiv.org/abs/1812.06110>.
- [12] Jaedeug Choi and Kee-Eung Kim. “Map inference for bayesian inverse reinforcement learning.” In: *Advances in Neural Information Processing Systems*. 2011, pp. 1989–1997.
- [13] Sungjoon Choi, Kyungjae Lee, and Songhwai Oh. “Robust learning from demonstrations with mixed qualities using leveraged gaussian processes.” In: *IEEE Transactions on Robotics* 35.3 (2019), pp. 564–576.
- [14] Adam Coates, Pieter Abbeel, and Andrew Y Ng. “Apprenticeship learning for helicopter control.” In: *Communications of the ACM* 52.7 (2009), pp. 97–105.
- [15] Yan Duan et al. “Benchmarking deep reinforcement learning for continuous control.” In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.
- [16] Tom Everitt and Marcus Hutter. *Avoiding Wireheading with Value Reinforcement Learning*. 2016. arXiv: [1605.03143v1](https://arxiv.org/abs/1605.03143v1) [cs.AI].
- [17] Tom Everitt, Gary Lea, and Marcus Hutter. *AGI Safety Literature Review*. 2018. arXiv: [1805.01109v2](https://arxiv.org/abs/1805.01109v2) [cs.AI].
- [18] Chelsea Finn, Sergey Levine, and Pieter Abbeel. *Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization*. 2016. arXiv: [1603.00448v3](https://arxiv.org/abs/1603.00448v3) [cs.LG].
- [19] Brendan Fong and David I Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 2019.
- [20] Vincent François-Lavet et al. *DeeR*. <https://deer.readthedocs.io/>. 2016.
- [21] Daniel H Grollman and Aude Billard. “Donut as I do: Learning from failed demonstrations.” In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3804–3809.
- [22] Robert Hecht-Nielsen. “Theory of the backpropagation neural network.” In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.
- [23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators.” In: *Neural networks* 2.5 (1989), pp. 359–366.

- [24] Jiayi Huang, Mostofa Patwary, and Gregory Diamos. *Coloring Big Graphs with AlphaGoZero*. 2019. arXiv: [1902.10162v2](https://arxiv.org/abs/1902.10162v2) [cs.AI].
- [25] Evan Hubinger et al. "Risks from Learned Optimization in Advanced Machine Learning Systems." In: *CoRR* abs/1906.01820 (2019). arXiv: [1906.01820](https://arxiv.org/abs/1906.01820). URL: <http://arxiv.org/abs/1906.01820>.
- [26] Marcus Hutter. *A Theory of Universal Artificial Intelligence based on Algorithmic Complexity*. 2000. arXiv: [cs/0004001v1](https://arxiv.org/abs/cs/0004001v1) [cs.AI].
- [27] Alex Irpan. *Deep Reinforcement Learning Doesn't Work Yet*. <https://www.alexirpan.com/2018/02/14/rl-hard.html>. 2018.
- [28] Joel Janai et al. "Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art." In: *arXiv preprint arXiv:1704.05519* (2017).
- [29] Julian Jara-Ettinger. "Theory of mind as inverse reinforcement learning." In: *Current Opinion in Behavioral Sciences* 29 (2019), pp. 105–110.
- [30] Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.
- [31] Karmeshu and N. R. Pal. "Uncertainty, Entropy and Maximum Entropy Principle — An Overview." In: *Studies in Fuzziness and Soft Computing* (2003), pp. 1–53. ISSN: 1860-0808. DOI: [10.1007/978-3-540-36212-8\\_1](https://doi.org/10.1007/978-3-540-36212-8_1). URL: [http://dx.doi.org/10.1007/978-3-540-36212-8\\_1](http://dx.doi.org/10.1007/978-3-540-36212-8_1).
- [32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.
- [33] Jianhua Lin. "Divergence measures based on the Shannon entropy." In: *IEEE Transactions on Information theory* 37.1 (1991), pp. 145–151.
- [34] Arushi Majha, Sayan Sarkar, and Davide Zagami. "Categorizing Wireheading in Partially Embedded Agents." In: *Artificial Intelligence Safety 2019, AISafety@IJCAI 2019, Macao, China, August 11-12, 2019*. 2019. URL: [http://ceur-ws.org/Vol-2419/paper%5C\\_31.pdf](http://ceur-ws.org/Vol-2419/paper%5C_31.pdf).
- [35] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), p. 529.

- [36] Ahmed H. Qureshi, Byron Boots, and Michael C. Yip. *Adversarial Imitation via Variational Inverse Reinforcement Learning*. 2018. arXiv: [1809.06404v3](https://arxiv.org/abs/1809.06404) [cs.LG].
- [37] Deepak Ramachandran and Eyal Amir. "Bayesian Inverse Reinforcement Learning." In: *IJCAI*. Vol. 7. 2007, pp. 2586–2591.
- [38] Eric Sven Ristad and Peter N Yianilos. "Learning string-edit distance." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.5 (1998), pp. 522–532.
- [39] Stuart J. Russell. *Human compatible: artificial intelligence and the problem of control*. Viking, 2019.
- [40] Daniel Salvadori. *huskarl*. <https://github.com/danaugrs/huskarl>. 2019.
- [41] Davide Secchi. "Bounded Rationality." In: *Extendable Rationality* (Sept. 2010), pp. 19–25. DOI: [10.1007/978-1-4419-7542-3\\_3](https://doi.org/10.1007/978-1-4419-7542-3_3). URL: [http://dx.doi.org/10.1007/978-1-4419-7542-3\\_3](http://dx.doi.org/10.1007/978-1-4419-7542-3_3).
- [42] Andrew Senior, John Jumper, and Demis Hassabis. "AlphaFold: Using AI for scientific discovery." In: *DeepMind* (2018). URL: <https://deepmind.com/blog/alphafold>.
- [43] Helene Seyr and Michael Muskulus. "Use of Markov Decision Processes in the Evaluation of Corrective Maintenance Scheduling Policies for Offshore Wind Farms." In: *Energies* 12.15 (2019), p. 2993.
- [44] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: [1712.01815v1](https://arxiv.org/abs/1712.01815) [cs.AI].
- [45] Herbert A Simon. "Theories of bounded rationality." In: *Decision and organization* 1.1 (1972), pp. 161–176.
- [46] Peter Sunehag and Marcus Hutter. *Principles of Solomonoff Induction and AIXI*. 2011. arXiv: [1111.6117v1](https://arxiv.org/abs/1111.6117) [cs.AI].
- [47] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [48] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers." In: *Neural processing letters* 9.3 (1999), pp. 293–300.

- [49] Amos Tversky and Daniel Kahneman. "Judgment under uncertainty: Heuristics and biases." In: *Judgment under uncertainty* (). Ed. by Daniel Kahneman, Paul Slovic, and Amos Tversky, pp. 3–20. DOI: [10.1017/cbo9780511809477.002](https://doi.org/10.1017/cbo9780511809477.002). URL: <http://dx.doi.org/10.1017/cbo9780511809477.002>.
- [50] Laura Graesser Wah Loon Keng. *SLM Lab*. <https://github.com/kengz/SLM-Lab>. 2017.
- [51] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. "Maximum entropy deep inverse reinforcement learning." In: *arXiv preprint arXiv:1507.04888* (2015).
- [52] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. "Modeling interaction via the principle of maximum causal entropy." In: (2010).
- [53] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.
- [54] Brian D Ziebart et al. "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior." In: *Proceedings of the 10th international conference on Ubiquitous computing*. ACM. 2008, pp. 322–331.
- [55] Brian D Ziebart et al. "Planning-based prediction for pedestrians." In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 3931–3936.





## COLOPHON

This document was typeset on a modified version of the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst’s seminal book on typography “*The Elements of Typographic Style*”. The typography on Aaron Turon’s PhD thesis has also influenced the design choices significantly.

It had been written on both Emacs `Org-mode` and  $\text{\LaTeX}$  and was converted using `pandoc`.

*Final Version* as of June 30, 2020 ¶.