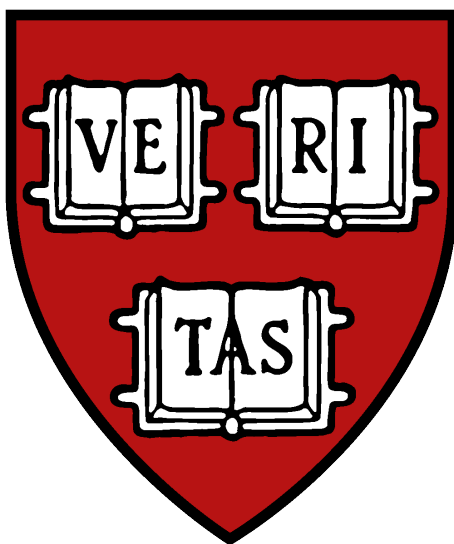


Imitation Learning Through Ranked Demonstrations

YI LIN WANG



Advised by David Parkes

A thesis presented in partial fulfillment
of the requirements for the degree of
Bachelor of Arts in Computer Science

Harvard College
Cambridge, Massachusetts
March 2021

Abstract

Much work has been done on *imitation learning*, in which an agent learns to mimic the behaviors of an expert. Work has also been done in *inverse reinforcement learning*, where an expert with a goal demonstrates behaviors to an agent, who infers a reward function describing the goal that these behaviors are meant to achieve. An agent then learns to achieve the intended goal of the expert by using regular reinforcement learning on the learned reward function. In both imitation learning and inverse reinforcement learning, much attention has been directed at using human demonstrations where a behavior is performed optimally (i.e. without mistake) as the sole input for agents to learn human intentions from.

We employ recent work on using ranked suboptimal human demonstrations, where the demonstrations do not perfectly represent a desired behavior, in order to infer human intentions. We use this as a method of imitation learning, and explore the use of this method in the Atari game Space Invaders to train agents to exhibit new behaviors. Typically, imitation learning in this environment involves trying to mimic an expert who is good at playing the game. We attempt to learn behaviors that are different from the standard objective of maximizing game score, and do not have an associated reward function for determining good or bad performance that has already been specified for us. This work fits into the broader context of helping agents learn to act in environments where it is difficult to specify a reward function with which to train an agent.

Acknowledgements

I thank David Parkes for wonderful advice and guidance over the last year. I thank my concentration adviser Yiling Chen for being a reader. I thank Matthias Gerstgrasser and Gianluca Brero for providing support and feedback on research last summer. I thank Jason Ma, Brian Chu, David Zhu, and Jeff Jiang for providing a significant amount of help in setting up my project and providing feedback.

I thank my blockmates of four years: Siva Emani, Zach Fraley, Robert Jomar Malate, David Moon, and Aidan Stoddart, who have provided me with support through all the ups and downs of college life since I set foot on campus as a first year student. Additional thanks to Sophie Edouard, Christina Tran, Rachelle Ambroise, Audrey Pettner, Chloe Saracco, Freddie Shanel, and Ian McGregor for being amazing and loving linkmates, an unusual part of the social experience here. Thanks to Antares Tobelem and Lara Teich for being part of our social circle. Special thanks to Ami Ishikawa for having breakfast with me so many times over the last several years.

Shoutout to Zoom Tech Squad. I also owe Dylan Li, William Yao, Claire Shi, Vincent Li, Kim Nguyen, Seeam Shahid Noor, Diego Arias, Josh Mathews, Johannes Lang, Richard Muratore, Stephen Nyarko, Taras Holovko, Ralph Estantboulieh, Anna Lou, Karen Chen, Ava Jiang, Ryan Lee, Sabrina Chern, Edward Zhao, Aditya Dhar, Clarence Chan, Bryan Lee, and Jonathan Chu for many insightful, serious, trivial, and hilarious conversations in the dining hall and in quarantine.

There are countless others who have given me much both in college and throughout my life who I appreciate, and I could fill many pages with their names. They are all meaningful to me.

Finally, I want to express love and appreciation to both of my parents for bringing me into the world and raising me to be the person I am.

Contents

Abstract	iii
Acknowledgements	iv
Contents	v
List of Figures	vii
Chapter 1 Introduction	9
1.1 Contributions	11
1.2 Related Work	12
Chapter 2 Reinforcement Learning	14
2.1 Markov Decision Processes	14
2.2 Policy Gradient	18
2.3 Q-Learning	19
2.4 Proximal Policy Optimization	20
2.5 Summary	22
Chapter 3 Imitation Learning	23
3.1 Motivation	23
3.2 Behavior Cloning	24
3.3 Compounding Error	25
3.4 Summary	28
Chapter 4 Inverse Reinforcement Learning	29
4.1 Motivation	29
4.2 Feature Matching	30
4.3 Maximum Entropy Methods	32

4.4	Adversarial IRL	34
4.5	Discussion	36
4.6	Reward Extrapolation	37
4.7	Summary	40
Chapter 5	Trajectory Ranked Imitation Learning	41
5.1	Method	41
5.2	Experiments	43
5.3	Discussion	48
5.4	Summary	49
Chapter 6	Interpretation of Reward Functions	50
6.1	Method	50
6.2	Results	51
6.3	Summary	59
Chapter 7	Conclusion	60
7.1	Further Work	61
Appendices		64
A	Experiment Setup	64
B	Convolutional Neural Networks	65
C	TRIL Demonstrations	66
Bibliography		75

List of Figures

4.1 The T-REX approach of learning a reward function, and then training an agent using that reward function [Brown et al. 2019].	39
4.2 Game score reward extrapolation [Brown et al. 2019].	40
5.1 Shots fired reward extrapolation graph produced by our own experiment.	42
5.2 An agent only firing at column 1.	44
6.1 Column 1 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.	52
6.2 Column 2 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.	53
6.3 Column 3 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.	54
6.4 Column 4 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.	55
6.5 Column 5 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.	56
6.6 Column 6 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.	57

6.7 Row 1 attention maps above and frame images below, for 4 different states.

The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

CHAPTER 1

Introduction

In the study of *artificial intelligence* and *machine learning*, we ask the question of how to build intelligent systems [Russell and Norvig 2020]. Generally we think of intelligent systems as being able to make observations about their environment, and make complex decisions.

Deep learning allows us to employ complex models to describe the world and take actions, by allowing us to quickly find parameters in a large parameter space that make these complex models effective. Deep learning is great for perception and observation because it leverages large amounts of data to help agents make observations and predictions, and the key advance is that it often allows us to work end-to-end, from observation directly to action [LeCun et al. 2015]. Previously, without recent deep learning advances, feature extraction had to be manually implemented, with outputs as the result of many manually constructed parts put together. With end-to-end deep learning, agents in an environment can learn appropriate features that are useful directly from the data, without manual construction.

Reinforcement learning (abbreviated RL) is a method to mathematically formalize the problem of agents taking actions, making observations about the consequences of those actions, and receiving rewards or punishments for their actions [Sutton and Barto 2018]. Therefore it deals with the decision making part of intelligence. Standard RL in the past ran into problems with feature engineering, where it was difficult to think of all the different features of the environment that might be relevant to help an agent learn a good *policy*, which is a plan for how to act in the world.

By employing the tools of deep learning, *deep reinforcement learning* resolves this issue by making this process end-to-end as well, so that agents can learn features that help them make good decisions, and perhaps even learn better features than ones that humans could have engineered for them to learn, such as in the case of Atari games [Mnih et al. 2015], classic board games such as Go [Silver et al. 2016], and continuous control in robot manipulation [Schulman et al. 2015a].

A different paradigm that we will focus our attention on in this thesis is that of *imitation learning* (IL). Rather than specifying reward functions for agents to maximize, we provide demonstrations of how we want an agent to act in the world, and then ask the agent to mimic our behaviors as well as possible [Ross and Bagnell 2010]. In this setting, an agent attempts to learn a policy in which it takes actions that deviate as little as possible from the actions that would be taken from a given expert at each state in the environment. Imitation learning approaches have been effective in situations where it is difficult to define reward functions to learn to optimize, such as in driving tasks where an agent can simply learn to emulate an expert driver.

In reinforcement learning, there is also the problem of where rewards come from, and how they are defined for our agents. For environments like Atari games, the reward might be simple: we might just define the reward received by an agent to be the score in the game that the agent achieves. However, as discussed regarding imitation learning, in real world environments, the desired reward function might be very complicated and involve many different factors in conjunction with each other, such as the aforementioned driving example.

This is where we might want to have an agent infer a reward function by observing demonstrations provided by humans, which is often called *inverse reinforcement learning* (IRL) [Ng and Russell 2000]. IRL can play a role in helping us train agents to perform specific tasks. Suppose we intend to achieve a particular task, and demonstrate ourselves fulfilling the task. If an agent observes our actions, even if these actions do not perfectly achieve our intended goal, it might be able to infer what the intended goal is. Then, upon capturing the intended goal in the form of a learned reward function, the agent

can find a better solution for achieving the goal than in the demonstrations that we gave the agent. We often study IRL in situations where there is a notion of exceeding human performance that can be achieved, whereas in IL we simply want agents to mimic a desired behavior, often that of a human.

This thesis is structured as follows. Chapters 2-4 will provide background, while chapters 5-6 will provide our contributions. We will explore popular approaches to deep reinforcement learning (chapter 2), imitation learning (chapter 3), and inverse reinforcement learning (chapter 4) in the literature, and reflect on problems shared by the state-of-the-art in IL and IRL. In particular, we notice work in these fields is primarily done using “optimal” human demonstrations in order to infer the intentions of experts, which relies on humans being able to perform a desired behavior optimally, an assumption that does not always hold true. Furthermore, we notice that IL methods may require human intervention during training time to provide more demonstration data, and that IRL methods are often computationally intensive to implement. We turn our attention towards an IRL approach that uses suboptimal human demonstrations as a means of inferring human intentions, allows humans to only provide demonstration data once at the beginning of the training process, and is less computationally intensive. We present an adaptation of this approach to imitation learning with experiments (chapter 5), and then provide further analysis (chapter 6).

1.1 Contributions

We propose an algorithm called Trajectory Ranked Imitation Learning (TRIL) that employs recent work on using ranked suboptimal human demonstrations to infer human intentions [Brown et al. 2019] as a method of imitation learning. This recent work only studies ranked human demonstrations for IRL, not imitation, and only applies these techniques to behaviors that already have a reward function determining good or bad performance, such as maximizing game score in Atari games. We explore the use of TRIL in the Atari game Space Invaders to learn to mimic various expert behaviors that

do not have a natural reward function, and are difficult to manually write a reward function for in the game environment.

We apply this algorithm to multiple behavior types in Atari Space Invaders, which is a game involving a 6×6 grid of aliens to shoot. For each behavior type, we attempt to hit the aliens in a particular column or row instead of hitting as many aliens as possible (the standard way of playing the game). We submit demonstrations of each behavior to TRIL to learn reward functions, and we observe for several behaviors that applying RL on the learned reward function leads to good imitator performance. We show that our trained imitators display qualitatively better performance than behavior cloning methods that simply use supervised learning on optimal expert trajectories. Furthermore, we investigate properties of reward functions learned through ranked demonstrations by looking at game states that are assigned high or low reward. We provide evidence that ranked demonstration methods are capable of producing reward functions that capture desired features across an assortment of different behaviors.

Previous work in imitation learning and reward learning has largely focused on the game score as the ground truth reward function in Atari environments [Brantley et al. 2020] [Reddy et al. 2019] [Brown et al. 2019]. To the best of our knowledge, this is the first set of results in the literature in training reward-based imitators to exhibit more complex behaviors in Space Invaders than maximizing game score.

1.2 Related Work

This work fits into the broader context of helping agents learn to act in environments where it is difficult to specify a reward function with which to train the agent. This has often come in the form of imitation learning and inverse reinforcement learning, as mentioned above. An example of this is driving; it is difficult to write an explicit reward function for good or bad driving, so learning from human demonstrations has been a popular method [Codevilla et al. 2019] [Pan et al. 2017] [Zhang and Cho 2016]. Even in scenarios where it is possible to write explicit reward functions to evaluate performance,

it is sometimes much easier to mimic expert demonstrations than to try many different actions through exploration, such as in robotic arm manipulation [Akkaya et al. 2019]. Recent work on imitation includes an approach where the imitator first develops a model from exploration before learning from expert demonstrations [Torabi et al. 2018] and there has been work in learning to imitate behaviors from video demonstrations [Sermanet et al. 2018] [Liu et al. 2018]. Methods of learning to perform a task from a single demonstration have also been proposed [Yu et al. 2018] [Goo and Niekum 2019] but these approaches generally require extra training to achieve good performance on very similar tasks.

Imitation for its own sake is often an important goal of imitation learning, such as in human-AI collaboration tasks, where an agent attempts to actually model human behavior [Hu et al. 2020] [Carroll et al. 2019] rather than just using expert demonstrations as a method of learning to perform a particular task.

This fits into work in crowdsourcing human demonstrations in the loop of the training process to learn behaviors, such as work in reinforcement learning from human preferences [Christiano et al. 2017] and learning from rankings over demonstrations [Ibarz et al. 2018] that adapts work on deep learning over demonstrations [Hester et al. 2017]. In this work, human intervention is often required to rate the behaviors an agent is producing throughout the process of training the agent, which can often be time intensive for a human. For our imitation learning procedures, we instead apply a method of reward learning that does not require human intervention during training. We submit information about preferences over a wide range of the state space to begin with, instead of adaptively doing so during training time. This is desirable because it drastically reduces the amount of time that humans have to spend providing data to train an imitator.

CHAPTER 2

Reinforcement Learning

In this chapter, we will formalize the basics of reinforcement learning, building up to a description of proximal policy optimization (PPO), the RL algorithm that we will be using to train agents in our experiments.

We are starting here because we will be applying reinforcement learning techniques for our work in imitation learning, with the assumption that our environments are standard RL environments. We will also use techniques from inverse reinforcement learning, and in order to understand these techniques, it is important to first understand RL.

2.1 Markov Decision Processes

Crucial to reinforcement learning is the idea of a *Markov decision process* (MDP), which formalizes the environment that an agent lives in [Sutton and Barto 2018]. We assume that we have a set of states \mathcal{S} in the environment and a set of possible actions \mathcal{A} that an agent can take.

DEFINITION 1. A **Markov decision process** [Puterman 2014] contains

- A state space \mathcal{S} containing states s .
- An action set \mathcal{A} containing actions a .
- A reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that describes how good an action is for a given state.

- *Transition probabilities $p(s'|s, a) \in [0, 1]$ that describe the probability of moving to state s' given that action a is taken in state s .*

In the typical formulation of an MDP, we are able to directly observe the states in an environment. These are called *fully observed* MDPs, and for our purposes, we will just assume that MDPs are fully observed.

One way to think of an MDP is that we begin with a Markov chain consisting of states \mathcal{S} and a transition operator \mathcal{T} which is a matrix that defines the transition probabilities from one state to another.

An MDP expands on a Markov chain, giving it an action space \mathcal{A} and reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The transition operator is now a tensor (in this context, a three dimensional array of numbers) because the transition probability of moving to state s_{t+1} at time $t + 1$ depends on both the current state s_t and action taken a_t .

We imagine the agent as traveling along states in an MDP by taking actions, where actions at certain states lead the agent to other states, collecting rewards associated with state-action pairs. We endow the agent in the environment with a *policy* that describes the actions that are taken by the agent in any particular state that the agent is in.

This policy can be deterministic, in the sense that the agent always takes one specific action for each state, and in this case the policy is a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$. However, policies are often nondeterministic, and are represented by a probability distribution $\pi(a|s)$ over actions at each state s . In particular, we have that for each state s , $\sum_{a \in \mathcal{A}} \pi(a|s) = 1$. The deterministic case is just a special case of this where $\pi(a|s)$ is always 1 for the action taken at state s and 0 for all other actions.

In training an agent's policy, we often have parameters ϑ that determine the agent's policy. We update the policy by updating parameters ϑ . We thus often use the notation $\pi_{\vartheta}(a|s)$ to describe a policy with parameters ϑ .

The objective in RL is to find a policy $\pi_\vartheta(a|s)$ that maximizes the total reward received by an agent traveling along a trajectory in an MDP [Sutton and Barto 2018]. A finite *trajectory* with T time steps is a path along the MDP that specifies the states traveled to and actions taken by the agent at each of T different times, and is represented by $\tau = (s_1, a_1, \dots, s_T, a_T)$.

For a particular policy π_ϑ , we have a joint distribution over all possible trajectories given by

$$p_\vartheta(\tau) = p_\vartheta(s_1) \prod_{i=1}^T \pi_\vartheta(a_i|s_i) p_\vartheta(s_{i+1}|s_i, a_i),$$

where $p_\vartheta(s_{t+1}|s_t, a_t)$ is the probability of landing in state s_{t+1} given that we are in state s_t and take action a_t . We are thus trying to find a policy where, over all possible trajectories the agent could take, we receive the largest total reward in expectation.

Therefore, the objective of finding a policy maximizing total rewards is the objective of approximating

$$\vartheta^* = \arg \max_{\vartheta} \mathbb{E}_{\tau \sim p_\vartheta(\tau)} \sum_{t=1}^T r(s_t, a_t).$$

However, trajectories need not be finite. For an infinitely long trajectory, we make an assumption that there exists a stationary distribution $p_\vartheta(s, a)$ for the underlying Markov chain, where we now treat each state-action pair corresponding to a single time step as a single state of the chain. This distribution exists under some mild assumptions like *ergodicity*, where each state is reachable from any other state. The idea is that as we apply the transition operator \mathcal{T} to any starting distribution over state-action pairs in this Markov chain, we will eventually converge to the stationary distribution. Then in this case, we try to find

$$\vartheta^* = \arg \max_{\vartheta} \mathbb{E}_{(s,a) \sim p_\vartheta(s,a)} \frac{1}{T} \sum_{t=1}^T r(s_t, a_t),$$

where we include the $\frac{1}{T}$ factor to ensure finite total rewards.

In RL we generally care only about expectations over trajectories, not specific trajectories. So for example, even if our reward function is not differentiable, the expectation over policies of our reward might be differentiable, which is convenient for optimization over a function of this expectation.

The anatomy of an RL algorithm is typically as follows.

- (1) Sample generation. This is where a policy π_{ϑ} is run, and interacts with the environment, which collects data. We need this to get a sense of how good π_{ϑ} is at achieving our goals.
- (2) Model selection/reward calculation. This is where we might compute the expected reward given by the policy that was run, over the trajectories τ and timesteps t : $J(\vartheta) = \mathbb{E}_{\tau}[\sum_t r_t]$.
- (3) Policy improvement. We might do this through something like gradient ascent $\vartheta \leftarrow \vartheta + \alpha \nabla J(\vartheta)$ or backpropagation in a neural network that represents the policy.

In the process of calculating expected reward over a trajectory, we often find it useful to define the Q (*quality*) *function* where $Q^{\pi}(s_t, a_t)$ represents the expected total future reward associated with taking action a_t at step s_t , assuming we follow policy π . Similarly, we define the *value function* $V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi}[Q^{\pi}(s_t, a_t)]$ to be the expected reward that we will get at state s_t when following policy π . The value function thus tells us how good our current state is, if we follow policy π [Sutton and Barto 2018].

RL algorithms approach policy improvement differently. Some use *policy gradients* by directly differentiating the expected reward function and using gradient ascent and some estimate the Q function or value function and try to improve these without caring too much about the specific policy that these correspond to at each update. For example, for a given state s , after estimating $Q(s, a)$ for various actions a , we might just choose a policy where only the action a with the highest Q value at state s is taken. Some algorithms like *actor-critic* methods use a combination of Q function estimation and policy gradients [Sutton and Barto 2018].

If T is infinite, this is called an infinite time horizon, and the problem becomes that our estimates for the value function corresponding to a policy with parameters ϑ , denoted by V_{ϑ}^{π} , might fail to converge to a finite value. A simple way to solve this problem is to assume that we care more about rewards in the short term than in the long term. Thus we add a discount factor $\gamma \in [0, 1]$ ($\gamma = 0.99$ often works well), and attempt to maximize $\mathbb{E}[r(s_t, a_t) + \gamma V(s_{t+1})]$. This makes our values for the value function converge. In the MDP model, this is equivalent to adding a death state that is reachable with probability $1 - \gamma$ from the other states and has reward 0, and modifying each transition probability of the original MDP to be multiplied by γ .

2.2 Policy Gradient

Policy gradient is essentially gradient ascent applied to RL. For our finite time horizon, recall that our objective is to find the policy with parameters ϑ that maximize $J(\vartheta) = \mathbb{E}_{\tau \sim p_{\vartheta}(\tau)} \sum_{t=1}^T r(s_t, a_t)$. First, before optimizing J , we would like a way to estimate the value $J(\vartheta)$ for given ϑ . This can be done by sampling: we sample N different trajectories according to the distribution $p_{\vartheta}(\tau)$ and then average over the rewards obtained along those trajectories:

$$J(\vartheta) \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}),$$

where i is an index over different samples and t is an index over time steps [Sutton et al. 1999].

For gradient ascent (recall that we are maximizing rewards), we are going to need to find the gradient $\nabla_{\vartheta} J(\vartheta)$. We have that $J(\vartheta) = \int p_{\vartheta}(\tau) r(\tau) d\tau$, where $r(\tau)$ refers to the total reward given by trajectory τ . Since the gradient is linear, we obtain $\nabla_{\vartheta} J(\vartheta) = \int \nabla_{\vartheta} p_{\vartheta}(\tau) r(\tau) d\tau$.

There is a useful identity that tells us that $\nabla_{\vartheta} p_{\vartheta}(\tau) = p_{\vartheta}(\tau) \nabla_{\vartheta} \log p_{\vartheta}(\tau)$. Then substituting above, we have $\nabla_{\vartheta} J(\vartheta) = \int \nabla_{\vartheta} p_{\vartheta}(\tau) \nabla_{\vartheta} \log p_{\vartheta}(\tau) r(\tau) d\tau = \mathbb{E}_{\tau \sim p_{\vartheta}(\tau)} \nabla_{\vartheta} \log p_{\vartheta}(\tau) r(\tau)$ by the definition of expectation.

Recall that $p_{\vartheta}(\tau) = p(s_1) \prod_{t=1}^T \pi_{\vartheta}(a_t|s_t)p(s_{t+1}|s_t, a_t)$. Since the gradient is linear, we have that

$$\nabla_{\vartheta} \log p_{\vartheta}(\tau) = \nabla_{\vartheta} \log p(s_1) + \sum_{t=1}^T \nabla_{\vartheta} \log \pi_{\vartheta}(a_t|s_t) + \sum_{t=1}^T \nabla_{\vartheta} \log p(s_{t+1}|s_t, a_t).$$

The first and third terms (the third term includes the sum) in the sum above are 0 because the transition probabilities are given by the MDP, not the choice of policy ϑ . The middle term can be approximated because we know the policy π given by ϑ . It can be approximated by once again sampling trajectories according to ϑ , and this time calculating $(\sum_{t=1}^T \nabla_{\vartheta} \log \pi_{\vartheta}(a_t|s_t))(\sum_{t=1}^T r(s_t, a_t))$ for each sample and averaging over samples [Williams 1992]. Thus we have

$$\nabla_{\vartheta} J(\vartheta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\vartheta} \log \pi_{\vartheta}(a_{i,t}|s_{i,t}) \right) \left(\sum_{t=1}^T r(s_{i,t}, a_{i,t}) \right).$$

We can then use our estimate of $\nabla_{\vartheta} J(\vartheta)$ in gradient ascent to set $\vartheta \leftarrow \vartheta + \alpha \nabla J(\vartheta)$ repeatedly. We can further optimize our gradient updates with a method such as Adam [Kingma and Ba 2014].

Intuitively, policy gradient is a trial-and-error method when we look at the equations above, and whenever we attempt to play out the current policy in training, it makes actions with good reward more likely when we update the policy, and actions with bad reward less likely.

2.3 Q-Learning

We briefly mention Q-learning because of its importance in the RL literature. Q-learning searches for a policy to maximize the Q function instead of optimizing for the objective. The Q function from earlier can be written as

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}[V^{\pi}(s')],$$

where s' is the next state after taking action a at state s . The optimal policy π^* is the one such that $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$ for all π, s, a . The Q function for this policy, often denoted Q^* , is given by the Bellman equation [Bellman 1954]

$$Q^*(s, a) = \mathbb{E}[r(s, a) + \gamma \max_{a'} Q^*(s', a')].$$

Q-learning is an iterative method to approximate Q^* using regression [Mnih et al. 2015] [Hessel et al. 2018]. Classically, this was achieved by having the learning agent execute its policy and updating the corresponding Q value of each state-action pair visited by the update rule

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

where α is a learning rate [Watkins and Dayan 1992].

However, recent work has been done in deep Q-learning with deep Q networks (DQN) that employ deep neural networks to approximate the Q function, which is helpful for tasks where the state or action space are continuous. DQN has been used successfully in Atari environments to play games at superhuman levels [Mnih et al. 2015].

2.4 Proximal Policy Optimization

The RL algorithm that we will use for training agents under a reward function in our own experiments will be proximal policy optimization (PPO) [Schulman et al. 2017]. This is a policy gradient algorithm that optimizes the objective function while preventing overshooting, by clipping the gradient to a constrained region.

One challenge with policy gradient methods that we discussed above is that they tend to be highly sensitive to the choice of step size in updating the policy. If the step size is too small, it tends to take a very long time to make progress, and if the step size is too large, noise becomes a significant factor and may lead to extreme drops in performance of the model.

In order to address some of these issues, PPO computes an update at each step to optimize an objective function while also minimizing deviance from the previous policy, in order to stabilize training. It builds on previous work in Trust Region Policy Optimization (TRPO) [Schulman et al. 2015b], where a “surrogate” objective function is maximized with a constraint on the maximum size of the policy update:

$$\max_{\vartheta} \hat{\mathbb{E}}_t \left[\frac{\pi_{\vartheta}(a_t|s_t)}{\pi_{\vartheta_{old}}(a_t|s_t)} \hat{A}_t \right] \text{ subject to } \hat{\mathbb{E}}_t [KL[\pi_{\vartheta_{old}}(\cdot|s_t), \pi_{\vartheta}(\cdot|s_t)]] \leq \delta.$$

Here, \hat{A}_t is the empirical estimate of the so-called “advantage” function $A(s, a) = Q(s, a) - V(s)$ for the policy at time step t . The constraint is a measure of KL-divergence between the two distributions representing the old and new policies.

Letting $r_t(\vartheta) = \frac{\pi_{\vartheta}(a_t|s_t)}{\pi_{\vartheta_{old}}(a_t|s_t)}$, this surrogate objective becomes $\mathbb{E}_t[r_t(\vartheta)\hat{A}_t]$. We call $r_t(\vartheta)$ a probability ratio, and note that $r(\vartheta_{old}) = 1$.

PPO looks for an unconstrained objective function that limits large policy updates, and uses the objective

$$L^{CLIP}(\vartheta) = \hat{\mathbb{E}}_t[\min(r_t(\vartheta)\hat{A}_t, \text{clip}(r_t(\vartheta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

where ϑ are the parameters for the policy, \hat{E}_t is the expectation over time steps in the training process, r_t is the probability ratio under the new and old policies, \hat{A}_t is the estimated advantage at time t , and ϵ is a hyperparameter that is typically in the range $[0.1, 0.2]$.

The first term inside the min is the TRPO surrogate objective function. The second term modifies the surrogate objective by clipping the probability ratio so that it does not move outside of the range $[1 - \epsilon, 1 + \epsilon]$. Then, the minimum of these two terms is taken, so that the final objective function is a lower bound on the unclipped objective, so that the change in probability ratio is ignored when it makes the objective improve, and included when it makes the objective worse. PPO is able to achieve many state-of-the-art results in Atari environments [Schulman et al. 2017].

For PPO and other RL algorithms, it is common practice to train an agent for a large number of time steps (e.g. 10 million), and take a snapshot of the agent after 500 thousand time steps and every 1 million time steps, and then take the best performing agent as the result. We will use this setup when we apply PPO in our experiments.

2.5 Summary

In this chapter, we establish basic aspects of reinforcement learning and PPO, the RL algorithm that we will ultimately use to train our agents in our experiments. In the next chapter, we will describe the goals of imitation learning, which our work attempts to fulfill, and some of the standard approaches to imitation learning.

CHAPTER 3

Imitation Learning

In this chapter, we describe the fundamentals of *imitation learning*, since our methods and results are ultimately meant to achieve the goals of imitation learning. We will discuss *behavior cloning* and observe a few problems with standard approaches to imitation learning that motivate our discussions in following chapters.

3.1 Motivation

While RL methods are powerful, they rely on the presence of a reward function to rate each part of the state space and associated actions as being good or bad. It is often difficult to construct certain types of reward functions in complex environments in order to actually employ RL methods. Sometimes reward functions are simple to specify, such as the score in Atari games when the goal is simply to train an agent to play the game well. But in the real world, reward functions can be extremely complicated. For example, if we want to train an agent to drive a vehicle, then what we consider as good driving may be a complicated function of speed, control, maneuvering with respect to other drivers, and so on. It may be difficult to manually design a reward function that captures all of the desired features of good driving, that is defined for all parts of the state space and all possible actions.

In such a case, we might want to just demonstrate what good driving looks like instead of having to construct a specific reward function ourselves. Then we might try to ask agents to develop policies that mimic the behavior in our demonstrations, and

generalize well outside of demonstration data. This is an example of the paradigm that we refer to as imitation learning.

3.2 Behavior Cloning

One method of imitation learning is called *behavior cloning*, which is essentially supervised learning in the context of RL policies [Pomerleau 1991]. The model is that we have an agent who makes observations o . Observations reveal partial information about the world. States s refer to the world as it actually is at a point in time. The agent has a policy π_{ϑ} with parameters ϑ that determines what actions a it will take given particular observations. In general, we refer to $\pi_{\vartheta}(a|o)$ as a probability distribution over actions given various observations. If an agent performs an action at a particular state, with some probability it will move to some other state.

A standard assumption is that the underlying states are Markovian, in the sense that the distribution over the next state depends on the current state and action taken, but is independent of all other previous states, given the current state. Note that this does not mean the observations are Markovian, because the same observation could be produced by multiple states. When we only work with the states, the environment is called *fully observed*, and when we work with observations, the environment called *partially observed*. For our purposes, we will work with fully observed states.

In order to train an agent to mimic an expert behavior, an expert performs various actions according to the desired behavior in the MDP environment and provides the resulting state-action paths as trajectories τ_1, \dots, τ_N , where each trajectory $\tau_i = (s_0, a_0, \dots, s_T, a_T)$ consists of states s , each of which is paired with the expert action a taken at that state (here, we refer to T , the number of total timesteps, as the *length* of the trajectory).

We then treat the set of state-action pairs of the form (s_t, a_t) in each expert trajectory as training data in supervised learning where states s are inputs and actions a are

outputs, and the expert actions a for each state s are considered ground truth labels. If the action space is discrete, behavioral cloning is a classification problem, and if the action space is continuous, behavioral cloning is a regression problem.

The objective is then to find the policy π_{ϑ} with parameters ϑ maximizing the log-likelihood of N state-action pairs drawn from the expert trajectories, or equivalently, minimizing the negative log-likelihood:

$$\vartheta^* = \arg \min_{\vartheta} \left[-\frac{1}{N} \sum_{t=1}^N \log \pi_{\vartheta}(a_t | s_t) \right].$$

Thus an intuition is that in standard formulations of behavior cloning, we attempt to evaluate the policy of the agent by calculating the error of agent policy in terms of mismatch with expert policies, and attempt to update the agent policy to reduce this mismatch. There are many possible ways to measure this mismatch other than what we have described, some of which provide more information about expert and imitator policies [Ma 2020].

3.3 Compounding Error

Unfortunately, oftentimes the training data that is provided from the expert comes from a different distribution over states than that of the test data, because the imitator will take different actions from the expert, leading it to observe different parts of the state space than what is given as training data by the expert, and the imitator may be unable to cope with the resulting uncertainty of how to interact in a unknown situation in the environment. This is called *distributional shift* [Quinonero-Candela et al. 2008].

One common, related problem that is encountered empirically with behavioral cloning is that of *compounding error*. An agent is trained to fit a training set that involves observations labeled with actions that would be taken by an expert. When the agent encounters an observation that is slightly different from the ones it observed in the training set, it may take an action that is slightly divergent from what an expert

would do, but this causes even further divergence of observations from what the agent observed on the training set, and so on. Eventually, the agent is observing states that are very different from what it saw in the training set, and therefore performs poorly.

We can upper bound the error of behavior cloning as follows. Assume we use a 0-1 cost function c that is activated when the agent decides to choose an action for a state that the expert would not have chosen, where the expert uses policy π^* . So we let

$$c(s, a) = \begin{cases} 0 & \text{if } a = \pi^*(s) \\ 1 & \text{otherwise} \end{cases}.$$

Furthermore, assume that for every state s in the training set, the probability of the agent making a mistake on the training set after training is at most ϵ : this just means the agent can memorize the training set sufficiently well. We denote this by $\pi_\theta(a \neq \pi^*(s)|s) \leq \epsilon$.

THEOREM 1 (Adapted from [Ross et al. 2011](#)). *Under the cost function c , and the assumption that $\pi_\theta(a \neq \pi^*(s)|s) \leq \epsilon$ for all states s in the training data, the expected cost of behavior cloning in T timesteps is $\mathcal{O}(\epsilon T^2)$.*

PROOF. We want to bound the expected total loss over T time steps, where the expectation is over the agent's policy. At time step t , assuming the agent has followed the expert policy over all previous steps, the agent has probability ϵ of diverging, after which it may incur a loss at every single time step for the remaining $T - t$ time steps in the worst case. Thus we have

$$\mathbb{E}\left[\sum_t (c(s_t, a_t))\right] \leq \epsilon T + (1 - \epsilon)(\epsilon(T - 1) + (1 - \epsilon)(\dots)) = \mathcal{O}(\epsilon T^2).$$

□

This is not great; it means that the expected error by the agent increases quadratically with the number of time steps in the worst case. It is often necessary to provide very large numbers of expert demonstrations that cover the whole state space to make BC work well [[Sasaki et al. 2019](#)] [[Jeon et al. 2018](#)].

An approach that addresses this issue is the DAgger (Dataset Aggregation) algorithm [Ross et al. 2011], an iterative algorithm in which an agent learns a policy from a training set of labeled observations by an expert, employs the policy, and then the set of observations that result (with the actions taken by the agent discarded) is manually labeled with new actions by the expert. Then the agent learns from this new set of labeled examples, and this process is repeated.

THEOREM 2 (Adapted from Ross et al. 2011). *With the cost function $c(s, a) = \begin{cases} 0 & \text{if } a = \pi^*(s) \\ 1 & \text{otherwise} \end{cases}$, the expected error of DAgger on the training set is $\mathcal{O}(\epsilon T)$.*

PROOF. At each iteration of DAgger where the expert is involved in providing training data, the training data is matched to the state space that the imitator is exploring. Thus, the distribution that the training set comes from approaches the distribution that will be observed by the agent when we run its policy: $p_{train}(s) \rightarrow p_\vartheta(s)$. Our assumption is now that for all states $s \sim p_{train}(s)$, the policy π_ϑ diverges from the expert by at most ϵ . Then since $p_{train}(s) \rightarrow p_\vartheta(s)$, at each time step the agent has ϵ probability of making an error with respect to the expert, for T time steps, so the expected error is $\mathcal{O}(\epsilon T)$. \square

Under the most basic formulation, behavior cloning is simple to implement. This method runs into problems with distributional shift and compounding error. However, modifications to address this problem such as DAgger involve a significant amount of expert (often human) intervention in the loop to provide extra training data to an imitator to match the distribution of states that it is observing while interacting with the MDP environment. We therefore seek methods of imitation learning that allow the imitator to understand relevant features of large parts of the action space, without requiring humans to be in the loop of the training.

3.4 Summary

In this chapter, we introduce standard methods of imitation learning such as behavior cloning, and mention some of the problems that these methods face. In the next chapter, we will discuss inverse RL as a different approach to learning behaviors without a reward function.

CHAPTER 4

Inverse Reinforcement Learning

In this chapter, we will discuss classic methods for *inverse reinforcement learning* (IRL) and motivations for state-of-the-art work in IRL, including what are known as *adversarial* methods. We will observe problems with many of these approaches and discuss a ranked trajectory approach to IRL that deals with many of these problems. We discuss these ideas now because we will eventually adapt the ranked trajectory approach to achieve imitation results.

4.1 Motivation

The idea in IRL is to use optimal control as a model of human behavior: humans have some target reward function, and then try to find a sequence of actions that optimize the reward function. We can invert this model so that we observe a sequence of actions that supposedly maximize some unknown reward function, and try to recover a reward function that would make sense given the observed actions [Ng and Russell 2000]. Of course, humans tend not to actually act completely optimally, so there are some issues with this model. We will address some of these issues later on.

Why should we care about learning rewards? If we think about the standard imitation learning model, we only copy the behavior of experts, and not necessarily their intent; there is no reasoning about the outcomes of actions, in general. However, one different way of interpreting the goal of imitating a human is to actually pay attention to imitating the intent of the expert. If we wish to understand the intent of a human and mimic their intentions, we may actually take very different actions from that of

the expert if we feel that they better optimize for the expert’s intentions. Learning a reward function for an agent to optimize offers the potential for the agent to generalize the expert’s behavior to unseen parts of the state space in the training data.

Another reason we should care about learning rewards is the impracticality of constructing certain types of reward functions in complex environments. This is related to the motivation behind imitation learning. However, with inverse reinforcement learning, instead of giving up on finding a reward function altogether like in standard formulations of behavior cloning, we attempt to learn such a reward function even if we ourselves are not able to manually construct the reward function for the agent.

Thus we attempt to infer reward functions from demonstrations. One immediate issue is that this is an underspecified problem, so that we may have many reward functions that explain the same observed behaviors. In Section 4.3 we will introduce a probabilistic model to add constraints to this problem to make it more well defined.

More formally, with both regular forward RL and inverse RL, we are assumed to be given states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$, and an underlying MDP with transition function $p(s'|s, a)$ which we may or may not know. In forward RL, we are given the reward function $r(s, a)$, and attempt to learn an optimal policy $\pi^*(a|s)$ that maximizes expected reward. In inverse RL, we assume we have some trajectories $\{\tau_i\}$ sampled from an optimal or near-optimal policy π^* , and learn a reward function $r_\psi(s, a)$, where ψ are the parameters of this reward function. We then use regular forward RL to learn a policy π^* to maximize the expected reward returned by this learned reward function [Ng and Russell 2000].

4.2 Feature Matching

What form do we choose for our reward function? A classic choice was a linear function $r_\psi(s, a) = \sum_i \psi_i f_i(s, a) = \vec{\psi}^\top f(s, a)$ [Abbeel and Ng 2004], where f_i essentially corresponds to features of states and actions that we like or do not like, and $\vec{\psi}$ is a vector

of parameters for the reward function with components ψ_i . In practice, we could have a more complicated reward function represented by a neural network with parameters ψ corresponding to the parameters of the neural network, and this is the type of reward function that we will use later on in our experiments.

However, we will first discuss what is called *feature matching IRL*. We first focus on linear reward functions. We try to find weights ψ for features f such that in expectation, the optimal policy π^{r_ψ} under reward r_ψ has the same features $f(s, a)$ as the optimal expert policy π^* . That is, we want $\mathbb{E}_{\pi^{r_\psi}}[f(s, a)] = \mathbb{E}_{\pi^*}[f(s, a)]$. Given some ψ , we can approximate the RHS by calculating the features of the expert data and averaging, and we can approximate the LHS by first finding a policy π^{r_ψ} using standard RL techniques, and computing these features.

This is still an underspecified problem, because there are often multiple reward functions that cause these feature expectations to match. We bring in an idea from support vector machines (SVM) in the context of classification problems, that when faced with multiple possible perfect decision boundaries, we pick the decision boundary that maximizes distance in some way between points of different classes. Here, we will attempt to find a reward function with parameters ψ such that the expected reward of the optimal policy π^* is as far away as possible from the expected reward of any other policy [Ratliff et al. 2006]. Thus our objective becomes

$$\max_{\psi, m} m \text{ s.t. } \psi^\top \mathbb{E}_{\pi^*}[f(s, a)] \geq \max_{\pi \in \Pi} \psi^\top \mathbb{E}_{\pi}[f(s, a)] + m,$$

where m represents a margin between the optimal policy and any other policy. Through algebraic manipulation similar to that used for SVMs, this is equivalent to the objective

$$\min_{\psi} \frac{1}{2} \|\psi\|^2 \text{ s.t. } \psi^\top \mathbb{E}_{\pi^*}[f(s, a)] \geq \max_{\pi \in \Pi} \psi^\top \mathbb{E}_{\pi}[f(s, a)] + 1.$$

The problem with this new constraint is that it removes our notion of a margin m that would decrease as policies π^* and π become more similar in some sense (or are

identical). So we instead use the constraint

$$\psi^\top \mathbb{E}_{\pi^*}[f(s, a)] \geq \max_{\pi \in \Pi} \psi^\top \mathbb{E}_\pi[f(s, a)] + D(\pi^*, \pi),$$

where D is some distance between policies, for example, the magnitude of difference in expected features, and D will represent our margin.

However, this is a constrained problem, which is not ideal for deep learning. Thus we will discuss probabilistic approaches to IRL as follows, that circumvent some of these issues.

4.3 Maximum Entropy Methods

A common model of suboptimal human behavior in an MDP includes optimality variables $\mathcal{O}_{1:T}$, where $\mathcal{O}_t = 1$ if the behavior at time t is optimal, and $\mathcal{O}_t = 0$ otherwise. We have states and actions linked by transition dynamics $p(s_{t+1}|s_t, a_t)$, as well as probability of optimality $p(\mathcal{O}_t | s_t, a_t) = \exp(r(s_t, a_t))$. The probability of observing a trajectory conditioned on optimality variables $\mathcal{O}_{1:T}$ is then

$$p(\tau | \mathcal{O}_{1:T}) = \frac{p(\tau, \mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})} \propto p(\tau) \prod_t \exp(r(s_t, a_t)) = p(\tau) \exp\left(\sum_t (r(s_t, a_t))\right),$$

where $p(\tau)$ refers to the probability of observing trajectory τ given the physics of the system. Under this model, the most optimal trajectories are most likely, and less optimal trajectories become exponentially less likely. We can find probabilities that make this true through inference [Kappen et al. 2009].

What we would like is to learn the reward function r in the above model. We denote our reward model with parameters ψ by r_ψ . We use maximum likelihood estimation (MLE): given an observed trajectory τ from (optimal) expert π^* , we want to find the reward function parameters ψ that make these observations most likely under the model.

So we want to solve the optimization problem

$$\max_{\psi} \frac{1}{N} \sum_{i=1}^N \log p(\tau_i | \mathcal{O}_{1:T}, \psi).$$

Since the physics of the model do not depend on our reward parameters, the probability we optimize is just $\exp(\sum_t (r(s_t, a_t)))$, and then we have to normalize so that our probabilities sum to 1. In particular, we have an objective

$$\max_{\psi} \frac{1}{N} \sum_{i=1}^N r_{\psi}(\tau_i) - \log Z,$$

where Z is called the partition function $\int p(\tau) \exp(r_{\psi}(\tau)) d\tau$.

Taking the gradient of our objective function, we obtain

$$\frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{Z} \int p(\tau) \exp(r_{\psi}(\tau)) \nabla_{\psi} r_{\psi}(\tau) d\tau$$

which can be conveniently written as the difference of two expectations:

$$\mathbb{E}_{\tau \sim \pi^*} [\nabla_{\psi} r_{\psi}(\tau_i)] - \mathbb{E}_{\tau \sim p(\tau | \mathcal{O}_{1:T}, \psi)} [\nabla_{\psi} r_{\psi}(\tau_i)].$$

The first term is easy to estimate from the expert samples, but the second term requires some work. Without going into too much detail, for the second term, we can compute so-called “forward” messages $\alpha(s_t)$ and “backward” messages $\beta(s_t, a_t)$, and letting $\mu_t = \beta(s_t, a_t)\alpha(s_t)$, corresponding to a probability of visiting a particular state-action pair, we can estimate the second term as $\sum_{t=1}^T \vec{\mu}_t^{\top} \nabla_{\psi} \vec{r}_{\psi}$, where $\vec{\mu}^{\top} = (\mu_1, \dots, \mu_T)$.

This approach is called the maximum entropy (MaxEnt) IRL algorithm [Ziebart et al. 2008]. After calculating the messages, it is possible to estimate the gradient using the expression above, and use gradient ascent on ψ to maximize reward. If the state space and action space are small, this tends to work well. This is a maximum entropy method because if the reward function takes on a linear form, it maximizes the learned policy’s randomness while still meeting the features of the expert reward function.

Now we discuss how to transfer these ideas to large continuous state-action spaces. First, if we assume that we do not have access to the probability of trajectories given optimality variables and ψ , then we cannot immediately estimate the second term of the gradient. So we first would have to calculate these probabilities using any maximum entropy RL method, and then we would sample from the obtained policy to obtain trajectories. We could then estimate the gradient of the objective as

$$\frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{M} \sum_{j=1}^M \nabla_{\psi} r_{\psi}(\tau_j),$$

where the second term comes from policy samples. A deep learning version of this approach is possible [Wulfmeier et al. 2015].

More efficiently than optimizing our policy every time we update our reward, we can just improve the policy slightly in each step using a trick called *importance sampling*. In particular, *guided cost learning* [Finn et al. 2016] uses this algorithm: given an initial policy and expert demonstrations, samples are generated using the policy to update the reward. The gradient for updating the reward is given by

$$\frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{\sum_j w_j} \sum_{j=1}^M w_j \nabla_{\psi} r_{\psi}(\tau_j),$$

with *importance sampling weights* $w_j = \frac{\exp(r_{\psi}(\tau_j))}{\pi(\tau_j)}$. The reward is updated, and then the policy π is improved slightly with respect to the updated reward. New samples are generated from the new policy, and the process is repeated.

4.4 Adversarial IRL

This problem setting looks like an adversarial game between the updated reward function and the policy we generate: when the reward is updated, our policy is shown to be further away from the desired expert, and when the policy is updated with respect to the new reward, it moves closer to being like the expert. Eventually, if the policy

behaves exactly like the expert, then they are indistinguishable. This looks a lot like the approach behind generative adversarial networks (GANs) [Goodfellow et al. 2014].

In this framework, we have a *generator* $p_\theta(x|z)$ that generates samples from random noise z , and some target demonstration data p^* , and we attempt to build a binary classifier D called a *discriminator* that is able to distinguish between the generator and the demonstration data, while training the generator to fool the discriminator. Thus if $D_\psi(x) = p_\psi(\text{target}|x)$, we want to solve for

$$\arg \max_{\psi} \frac{1}{N} \sum_{x \sim p^*(x)} \log D_\psi(x) + \frac{1}{M} \sum_{x \sim p_\theta} \log(1 - D_\psi(x))$$

with the discriminator, and we want to solve for

$$\arg \max_{\theta} \mathbb{E}_{x \sim p_\theta} [\log D_\psi(x)]$$

for the generator.

The generator behaves similarly to a policy generating samples, and the target data takes the role of the expert demonstrations. The reward function takes the role of the discriminator. It turns out the optimal discriminator has the form

$$D_\psi(\tau) = \frac{\frac{1}{Z} \exp(r(\tau))}{\prod_t \pi_\theta(a_t|s_t) + \frac{1}{Z} \exp(r(\tau))}.$$

To run IRL as a GAN, we take samples from policy π_θ and expert p^* , and optimize for $\psi \leftarrow \arg \max_{\psi} \mathbb{E}_{\tau \sim p^*} \log D_\psi(\tau) + \mathbb{E}_{\tau \sim \pi_\theta} \log(1 - D_\psi(\tau))$. Then we modify the policy so that it produces samples that are closer to the demonstration data, using gradient methods, approximating $\nabla_{\theta} \mathcal{L} \approx \frac{1}{M} \sum_{j=1}^M \nabla_{\theta} \log \pi_\theta(\tau_j) r_\psi(\tau_j)$ [Finn et al. 2016].

A generalization of this approach is adversarial IRL (AIRL) [Fu et al. 2018]. The authors show promising results, such as recovering a reward function where they can place limitations on their agent and have the agent still work. For example, the authors start with a four-legged ant in simulation, teach it to walk as the reward, then break two of its legs, and it is still able to walk after recovering the reward.

A similar approach applied to imitation learning that does not deal with rewards is generative adversarial imitation learning (GAIL) [Ho and Ermon 2016], in which the authors only use a regular binary classifier as a discriminator, and directly optimize the policy using

$$\nabla_{\vartheta} \mathcal{L} \approx \frac{1}{M} \sum_{j=1}^M \nabla_{\vartheta} \log \pi_{\vartheta}(\tau_j) \log D_{\psi}(\tau_j).$$

This is easier to set up, although the discriminator knows nothing about the desired reward. Guided cost learning and GAIL are essentially the same approach, just with different representations of D .

4.5 Discussion

Unfortunately, the process of training GANs tends to be unstable [Kodali et al. 2017] [Arjovsky et al. 2017] which makes GANs very difficult to train. In particular, methods like AIRL and GAIL which rely on GANs tend to be very difficult to implement because they often require an extensive search in order to find parameters for training that lead to good performance.

Two problems arise with all of the deep learning based IRL methods that have been described up until now. One issue is that these methods attempt to find reward functions that justify given expert demonstrations, but expert demonstrations may not always be able to capture the intention of an expert. For example, if a human acts as an expert and is unable to perform a very difficult task optimally, these IRL methods will find reward functions that justify suboptimal demonstrations but may not capture the behavior that the human intended.

Another important issue is that these algorithms involve a costly “inner loop” of policy optimization. In this loop, these methods estimate a reward function, then optimize a policy with respect to this reward function, and employ this policy in the environment to evaluate how good this policy is with respect to the expert demonstrations. Based on this evaluation, these methods attempt to estimate a new reward

function that better describes the expert behaviors, and then repeat this process over and over. The necessity of repeatedly improving the estimate of the reward function and policy causes many IRL methods to be computationally expensive.

We will thus discuss a method for IRL that leverages suboptimal demonstrations to learn the intent of an expert demonstrator and does not require repeated reward function optimization.

4.6 Reward Extrapolation

We discuss a recent approach in IRL without the use of optimal demonstrations called Trajectory-ranked Reward EXtrapolation (T-REX) [Brown et al. 2019]. T-REX is given suboptimal demonstrations $\tau_1 \prec \tau_2 \prec \dots \prec \tau_T$ which are ranked by preference order. It then uses inference to learn a reward function \hat{r}_ϑ that explains why the trajectories are ranked the way they are, assigning lower reward to less preferred demonstrations and higher reward to more preferred demonstrations. Then policy optimization is done once with respect to this reward function. This eliminates the inner loop of repeated reward and policy optimization.

In this setting, trajectories are now only given as sequences of states $\tau = (s_1, \dots, s_T)$, and reward functions $r : \mathcal{S} \rightarrow \mathbb{R}$ are only defined over states.

This reward inference works through learning a binary classifier with parameters ϑ that predicts which of two trajectories is more preferable.

Specifically, the form of the loss function for the reward learner is

$$\mathcal{L}(\vartheta) = \mathbb{E}_{\tau_i, \tau_j \sim \Pi} \left[\xi \left(P(\hat{J}_\vartheta(\tau_i) < \hat{J}_\vartheta(\tau_j)), \tau_i \prec \tau_j \right) \right],$$

where $\hat{J}_\vartheta(\tau) = \sum_t \hat{r}_\vartheta(s_t)$ gives the total reward obtained in trajectory $\tau = (s_1, \dots, s_T)$, P is the probability that a pair of trajectories is correctly ranked by the reward function, and ξ is a binary classifier loss function.

The probability P is estimated using a softmax-normalized distribution

$$P(\hat{J}_\vartheta(\tau_i) < \hat{J}_\vartheta(\tau_j)) \approx \frac{\exp \sum_{s \in \tau_j} \hat{r}_\vartheta(s)}{\exp \sum_{s \in \tau_i} \hat{r}_\vartheta(s) + \exp \sum_{s \in \tau_j} \hat{r}_\vartheta(s)},$$

and the choice for ξ is the cross entropy loss, so the entire loss function is

$$\mathcal{L}(\vartheta) = - \sum_{\tau_i \prec \tau_j} \log \frac{\exp \sum_{s \in \tau_j} \hat{r}_\vartheta(s)}{\exp \sum_{s \in \tau_i} \hat{r}_\vartheta(s) + \exp \sum_{s \in \tau_j} \hat{r}_\vartheta(s)}.$$

The loss function is inspired by the Bradley-Terry [Bradley and Terry 1952] and Luce-Shepard [Luce 2012] models of preferences.

In the authors’ experiments for Atari games, they train policies using PPO for different durations to generate suboptimal demonstrations. They take short subtrajectories from these demonstrations and give them the same ranking labels as the full trajectories in order to obtain more data for training the reward model. The labels are determined by the actual ground truth rewards obtained by the trajectories; if the game score achieved by τ_i is less than that achieved by τ_j , then τ_i is labeled as less preferable than τ_j .

Whenever they compare a subtrajectory of τ_i with a subtrajectory of τ_j where $\tau_i \prec \tau_j$, they make sure that the subtrajectory for τ_i does not begin after the subtrajectory of τ_j , so that they do not end up comparing the later part of a worse trajectory with the earlier part of a better trajectory; this may be an unfair comparison, as the later part of a worse trajectory may look like a higher reward state than an earlier part of a better trajectory. This also encourages the model to learn a reward function that is monotonically increasing over time, which works well for many Atari games. They hide the score in the state observations so that the reward model would have to rely on observations of gameplay to determine a reward function.

After training the reward model \hat{r}_ϑ , they train a policy with forward RL to learn a policy to maximize expected reward under \hat{r}_ϑ .

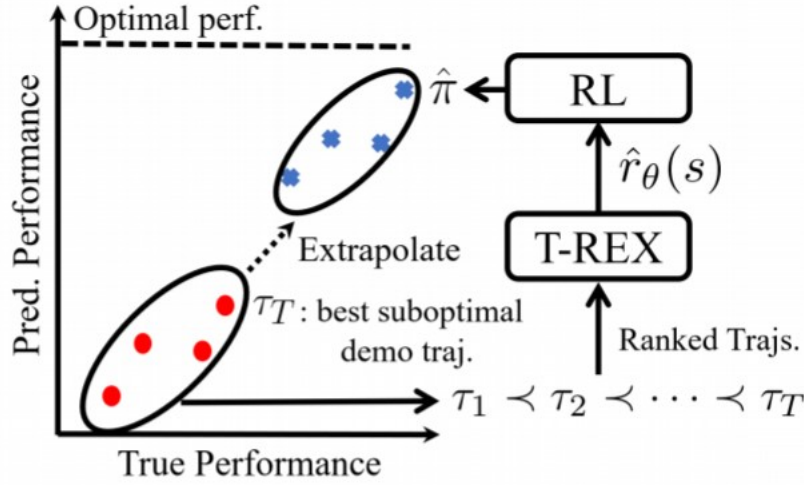


FIGURE 4.1: The T-REX approach of learning a reward function, and then training an agent using that reward function [Brown et al. 2019].

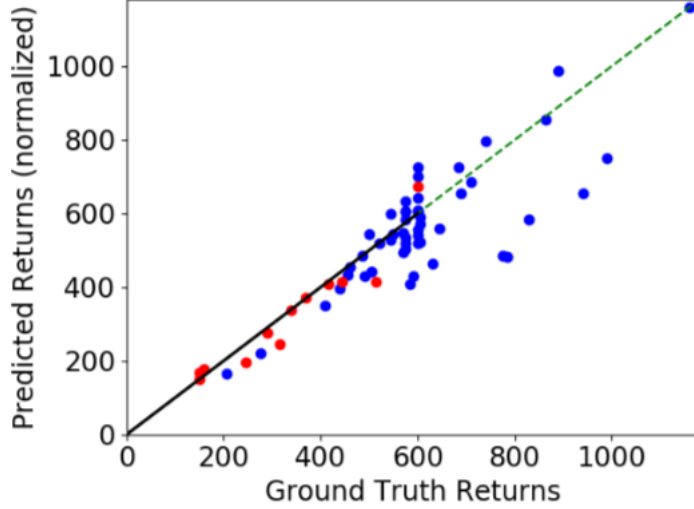
We use Space Invaders as our case study here. In this game, a player is allowed to move left and right at the bottom of a screen, and shoots at a grid of aliens who shoot at the player. The player has three lives, and loses a life every time they are hit by an alien. Their score is increased whenever they hit an alien with their shot, and more points are awarded for hitting aliens that are farther away from the player (closer to the top of the screen). In addition, there are obstacles near the bottom of the screen separating the player and the aliens; shots from both players and aliens erode these obstacles.

The authors obtain strong IRL results using T-REX; they are able to obtain policies whose average performance far exceeds that of the demonstrations for games like Space Invaders.

The authors also test their learned reward function against the true reward function (game score) in Space Invaders, and the results are shown in Figure 4.2.

In this figure, the vertical axis is the predicted game score by the learned reward function, and the horizontal axis is the actual game score. Each of the red dots represents a trajectory that is used to train the reward function, and the blue dots represent test

FIGURE 4.2: Game score reward extrapolation [Brown et al. 2019].



trajectories given to the learned reward function, asking it to predict its actual return. The reward function learned for Space Invaders seems to be able to extrapolate to trajectories with scores beyond those found in the training data and assign rewards that are similar to the actual game score.

Thus, T-REX is a promising approach to IRL that does not need optimal demonstrations to work well and does not utilize a computationally expensive loop alternating between reward and policy optimization.

4.7 Summary

In this chapter, we review classic algorithms for IRL, including adversarial methods. We discuss problems with these methods that are largely improved with an approach called T-REX involving ranked demonstrations. In the next chapter, we will propose an adaptation of this approach for imitation learning and present the results of using this method to imitate multiple types of expert behaviors in Space Invaders.

Trajectory Ranked Imitation Learning

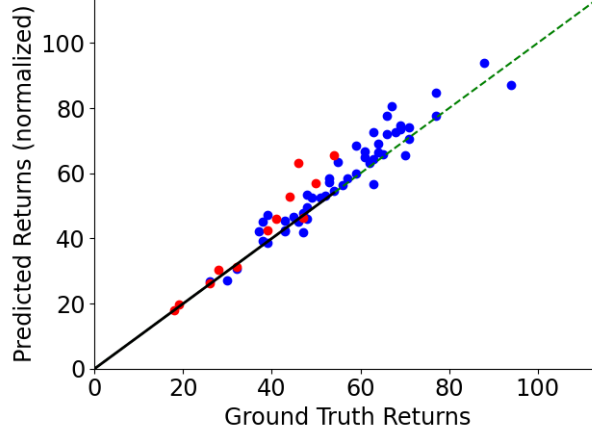
In this chapter, we motivate the adaptation of T-REX to learn more diverse reward functions for the imitation of behaviors. We present our algorithm, Trajectory Ranked Imitation Learning (TRIL), employ it on multiple challenging behaviors in Atari Space Invaders, and show that it outperforms behavior cloning in these settings.

5.1 Method

Given the success of T-REX on the standard reward function for Space Invaders, we ask if these methods of reward extrapolation work for different types of rewards. For example, one simple difference we can make is to generate agents whose reward is defined by how many shots they fire, independent of which aliens they hit. Under the environment with this reward, we use agents trained under PPO for different durations in order to generate various demonstrations, just as the authors of T-REX do. We use the same method of trajectory rankings to learn a reward function.

The results are in Figure 5.1. Just like in Figure 4.2, the vertical axis is the predicted score by the learned reward function, and the horizontal axis is the actual number of shots fired. Each of the red dots represents a trajectory that is used to train the reward function, and the blue dots represent test trajectories given to the learned reward function, asking it to predict its actual return. We see here that the learned reward function generally captures the desired feature of shots fired, and is able to extrapolate beyond the training set to associate observations correlated with shooting more with greater reward.

FIGURE 5.1: Shots fired reward extrapolation graph produced by our own experiment.



This is a relatively simple example, but a hypothesis is that we might be able to leverage this technique to learn reward functions that are very difficult to specify in the environment. For example, it is easy to manipulate the game environment so that each time a player fires a shot, reward increases, but it is less simple of a task to specify a reward function in which it is good to shoot specific aliens and not others.

In this situation, since we cannot train agents using a pre-existing reward function to provide demonstration data, we play the game ourselves to generate demonstrations.

We propose the following algorithm for imitation learning from ranked trajectories, called Trajectory Ranked Imitation Learning (TRIL). We define our reward network $\mathcal{R} : \mathcal{S} \rightarrow \mathbb{R}$, and we define the sum of the returns of the reward network on a trajectory $\tau_i = (s_1, \dots, s_T)$ as $\hat{\mathcal{R}}(\tau_i) = \sum_{t=1}^T \mathcal{R}(s_t)$.

Algorithm 1 Trajectory Ranked Imitation Learning (TRIL)

Input: Human created trajectories $\tau_1 \prec \dots \prec \tau_N$ where τ_N is an optimal demonstration of the desired expert behavior.

- (1) Train a reward network $\mathcal{R} : \mathcal{S} \rightarrow \mathbb{R}$ that maps any observed state in the environment to an associated reward.
 - (2) Compute $\hat{\mathcal{R}}(\tau_1), \dots, \hat{\mathcal{R}}(\tau_N)$ and check that $\hat{\mathcal{R}}(\tau_1) \lesssim \dots \lesssim \hat{\mathcal{R}}(\tau_N)$.
 - (3) Run PPO (or another reinforcement learning algorithm) using \mathcal{R} as a reward function on states to train an imitator.
-

The reward network is trained using the same loss function and method as in T-REX. In practice, we normalize the reward function by feeding its outputs to the sigmoid

function $S(x) = \frac{1}{1+e^{-x}}$. In our experiments with TRIL, we run PPO for 10 million time steps.

We want to check that there is an approximate ordering over rewards associated with the submitted demonstrations: $\hat{\mathcal{R}}(\tau_1) \lesssim \dots \lesssim \hat{\mathcal{R}}(\tau_N)$. We also may want to check that the accuracy of the reward function on ranking pairs of short subtrajectories is high (where subtrajectories have the same ranking labels as their full trajectories). If this is not the case, it is likely that the submitted rankings over trajectories are too confusing for the reward learner to extract meaningful features from, and that we ought to submit different trajectories, or different rankings over them.

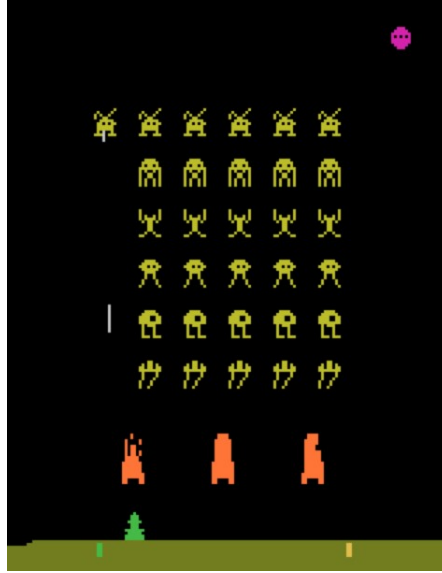
A key difference between T-REX and TRIL is that T-REX applies trajectory ranking based on *preexisting* ground truth reward functions (e.g. the game score when learning to play Space Invaders) to learn new reward functions that can be compared to the true reward function. In the setting of TRIL, there *is no* ground truth reward function that is explicitly defined; instead, we order demonstrations ourselves according to human submitted preferences that the reward learner attempts to explain using some mapping of states to reward values.

Furthermore, T-REX is a method for IRL that explicitly is meant to train agents that achieve *greater* rewards than the best demonstrations achieve on tasks. For the imitation framework that we work with in TRIL, there is no such thing as performance being “better” than the best demonstration, and the goal will be to mimic the best performance as well as possible.

5.2 Experiments

We set up a suite of more difficult tasks where we attempt to shoot at specific aliens and not others. For example, one of the behaviors we want to learn a reward function for is that of shooting the first column of aliens on the left in Space Invaders. Our convention is to call the leftmost column “column 1,” the second column from the left

FIGURE 5.2: An agent only firing at column 1.



“column 2,” and so on. The behavior of only shooting at column 1 is shown in Figure 5.2, and we will call this the “column 1 behavior.”





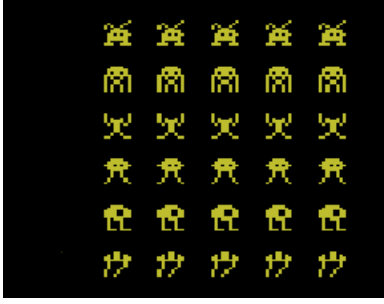

In our experiments using the procedure outlined in TRIL, we use 12 demonstrations to train each reward function. After training our imitators, we cap our environment so that trajectories end early, usually after a length of 300 steps, and in both training and in the test environment, we make the game end as soon as the agent loses a life (in regular Space Invaders, the agent gets 3 lives). We do this because the reward function is defined over states and is very sensitive to trajectories lasting longer; an agent can get larger total reward by staying alive in a low reward state, and the behaviors that we wish to imitate only require a single life to both demonstrate and imitate.


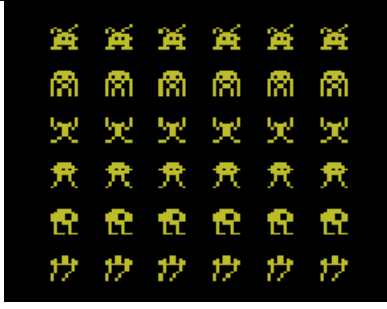

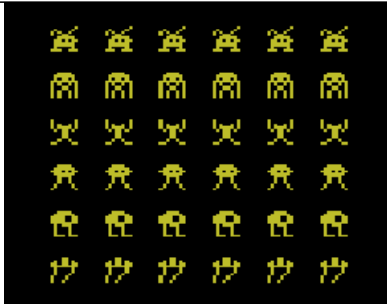

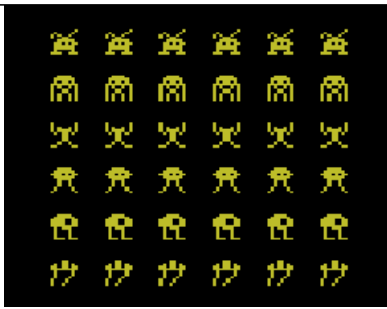

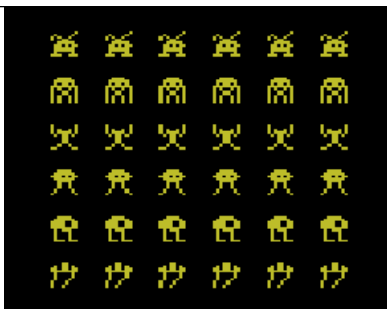

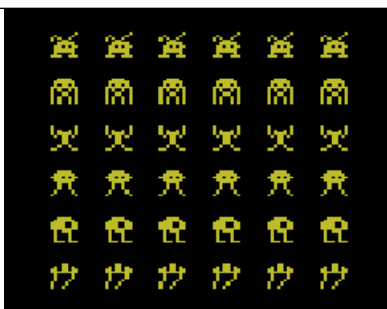
Space Invaders is a deterministic environment, so for any given “seed” of the environment the aliens move and shoot in predefined ways. So in our experiments we train over different seeds where the aliens shoot and move differently.

We attempt this same experiment with behavior cloning on 12 demonstrations of perfect performance (i.e. where in each demonstration we eliminate all of the correct

aliens and hit no others), so that with both TRIL and BC, we are using roughly the same amount of training data.

We apply this experiment setup to four different behaviors: shooting column 1, shooting column 2, shooting column 3, and shooting the bottom row of aliens (which we will refer to as row 1). We attempt each behavior across three seeds labeled 0, 1, and 2. We compare the aliens hit by trained imitators in both methods by providing an image of the aliens on screen remaining after letting each imitator play in the environment. The summary of the results across all behaviors are as follows. We provide details about the submitted demonstrations for TRIL in Appendix C.

Behavior	Seed	TRIL Targets Hit	BC Targets Hit	Winner
Shoot column 1	0			Tie
Shoot column 1	1			Tie
Shoot column 1	2			Tie

Shoot column 2	0			TRIL
Shoot column 2	1			TRIL
Shoot column 2	2			TRIL
Shoot column 3	0			TRIL
Shoot column 3	1			TRIL

Shoot column 3	2			Tie
Shoot row 1	0			TRIL
Shoot row 1	1			Tie
Shoot row 1	2			TRIL

We interpret these results and give commentary on the method in the following section.

5.3 Discussion

In our imitation results across different behaviors, we see that in terms of the aliens hit, TRIL achieves the same or better performance than BC on every task. Indeed, BC fails to learn a policy that shoots aliens in every task other than the column 1 behavior. The behaviors that are most suitable for TRIL are the ones where columns 1, 2, and 3 are targeted, and where row 1 is targeted.

While the TRIL imitators usually hit some stray aliens not in the desired row/column in the column 2, 3, and row 1 behaviors, in all cases where they are successful in hitting targets, they do not simply methodically clear all the aliens to do so (the usual way that strong agents play Space Invaders) and most stray aliens are in the leftmost column where the agent starts the game.

We attempted these experiments on behaviors where the columns 4, 5, and 6 must be individually hit; we did not achieve good imitation results with either TRIL or BC. Based on the behavior of trained imitators, we hypothesize that these tasks are very challenging because the agent in Space Invaders always begins on the left side of the screen, and must learn to move to a specific spot on the right side of the screen in order to hit these columns, while also learning to not shoot while on the left side of the screen. We see for instance in the column 3 and row 1 behaviors that it is very common for these imitators to eliminate the first column before moving to the right side of the screen to do something different.

In practice, when imitators are trained to imitate these tasks, they get stuck in local optima with respect to the reward function on the left side of the screen. Moving the agent to the right side of the screen using a trajectory ranked approach would likely require submitting more demonstrations that not only have gradually lower rank as they move toward the left side of the screen, but demonstrate the undesirability of shooting at various aliens in many different configurations on that side of the screen in order to sufficiently explore the state space.

In submitting a set of suboptimal trajectories to a reward learner, there are a couple of considerations at play. It is important to penalize behaviors that are undesirable, by ranking them low; for instance, in column shooting behaviors in Space Invaders, we choose to submit trajectories where we hit aliens in columns that we do not want to hit, and rank them lower in order to force the reward learner to penalize these states.

On the better end of the demonstrations, it is also useful to submit gradual improvements in demonstrations, for example orderings of trajectories where we hit one more correct alien in each trajectory than the one directly ranked underneath it. This allows the reward learner to more easily capture the relevant feature of the behavior, than if we simply provide multiple optimal demonstrations of the desired behavior and bad demonstrations of undesired states. See Appendix C for more details on our setup.

Some other benefits of TRIL over imitation learning methods that only use optimal human demonstrations include the leveraging of “bad” actions to explore undesirable state spaces and label them as being “worse” than good state spaces without having to manually give precise numerical reward values to these states. Furthermore, the benefit of learning a reward function is precisely that it is then possible for an agent to capture the “intent” of an expert, instead of simply trying to copy the exact movements of experts in their demonstrations, which can lead to overfitting. As we see in comparison to behavior cloning methods, we have instances where TRIL performs relatively well but BC does not achieve the desired behavior at all.

5.4 Summary

In this chapter, we propose an imitation learning algorithm called TRIL and observe that it outperforms behavior cloning at mimicking several difficult behaviors in Space Invaders. In the next chapter, we provide further evidence that learned reward functions produced by ranked demonstrations capture desired aspects of the intended behavior across different behavior types.

Interpretation of Reward Functions

In this chapter, we investigate features of reward functions across various behaviors that we learned using TRIL. We give evidence that ranked demonstrations produce reward functions that capture our intentions across these different behaviors.

6.1 Method

TRIL relies on RL algorithms performing well using the learned reward functions. But even in cases where RL algorithms do not perform well with respect to a learned reward function, we can still investigate whether the learned reward function adequately captures the desired behaviors, assigning high reward to areas of the state space corresponding to the behavior being performed well, and assigning low reward to areas of the state space corresponding to the behavior being performed poorly.

In particular, we look at actual in-game frames in the demonstration data that are given high reward by each reward function, in the hopes that they correspond to desirable states given our intentions.

We also adapt a method of generating “attention maps” for particular frames or states in the game that show where in a frame the reward model is paying the most attention [Greydanus et al. 2018]. To do this, we move a 3×3 (pixel) mask over each frame, and the mask color is set to be the default background color for the game. For each masked region, we compute the absolute difference in predicted reward when the region is not masked and when it is masked. The sum total of absolute changes in

reward for each pixel is used to generate a heat map that measures the influence of different regions of the image on the predicted reward. Pixels that are redder/brighter correspond to greater attention by the reward function and pixels that are bluer/darker correspond to less attention. We do this for the states associated with the maximum and minimum reward given to any state in the demonstrations by the reward function, and we also look at some other high reward states.

In particular, for each reward function, we sort all of the states in the training data by the reward associated to each state, and look at the states corresponding to the minimum, 90th percentile, 95th percentile, and maximum among rewards given to individual states. We generate an image of the game frame in each of these states, as well as an attention map. We observe that in the vast majority of cases, the high reward states correspond with good representations of the desired behavior.

6.2 Results

We generate attention maps and frames for reward functions trained on columns 1-6 behaviors, and the row 1 behavior, described in the previous chapter. We note that we do not achieve imitation results for column 4-6 behaviors, but we still find interesting results in the interpretation of learned reward functions for shooting these columns.

6.2.1 Column 1 Behavior

Figure 6.1 shows the frames and attention maps for the column 1 reward function. The state of the game assigned least reward is the one where the player is beginning to play at the left side of the screen, and all of the aliens are still remaining. The state assigned maximum reward is one where all of the aliens in the first column are eliminated, and no other aliens are hit, which is exactly what we want for optimal behavior. We also observe that when we look at the 95th percentile reward state, 4/6 aliens in column 1 are eliminated, and in the 90th percentile reward state, 2/6 aliens in column 1 are eliminated. This indicates that the learned reward function associates higher reward

to more aliens in column 1 being eliminated, which improves our confidence that the relevant feature of good performance is captured by this reward function.

The attention maps indicate that the reward function pays significant attention to aliens remaining in the first column when there are still aliens there, although the reward function does pay attention to a few aliens in the other columns. In particular, in the maximum reward state, most of the attention is shifted onto the aliens in column 2. We hypothesize that the learned reward function pays attention to aliens that might potentially shoot the player and end the game, as this significantly influences the total reward given to the player.

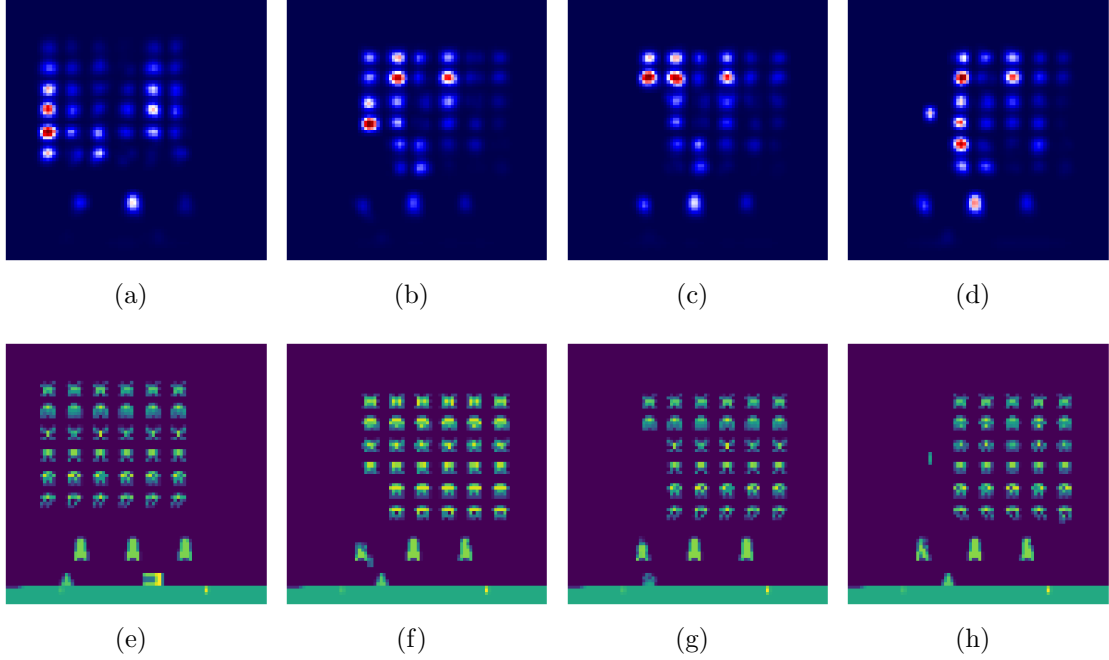


FIGURE 6.1: Column 1 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.2.2 Column 2 Behavior

Figure 6.2 shows the frames and attention maps for the column 2 reward function. We observe that no aliens are shot in the minimum reward state, and that all aliens in

column 2 are eliminated with no other aliens hit in the high reward (90th percentile, 95th percentile, maximum) states. The differences between high reward states do not provide as much information in this behavior since all of them essentially exhibit the optimal state according to our intentions, but it is promising that the high reward states are desirable states. With the attention maps, we see a decent amount of attention on the bottom aliens in column 1, perhaps indicating that these aliens are dangerous to the player, who is nearby. The prominence of other aliens in the attention maps is less interpretable but may also allude to aliens that pose a threat to the player.

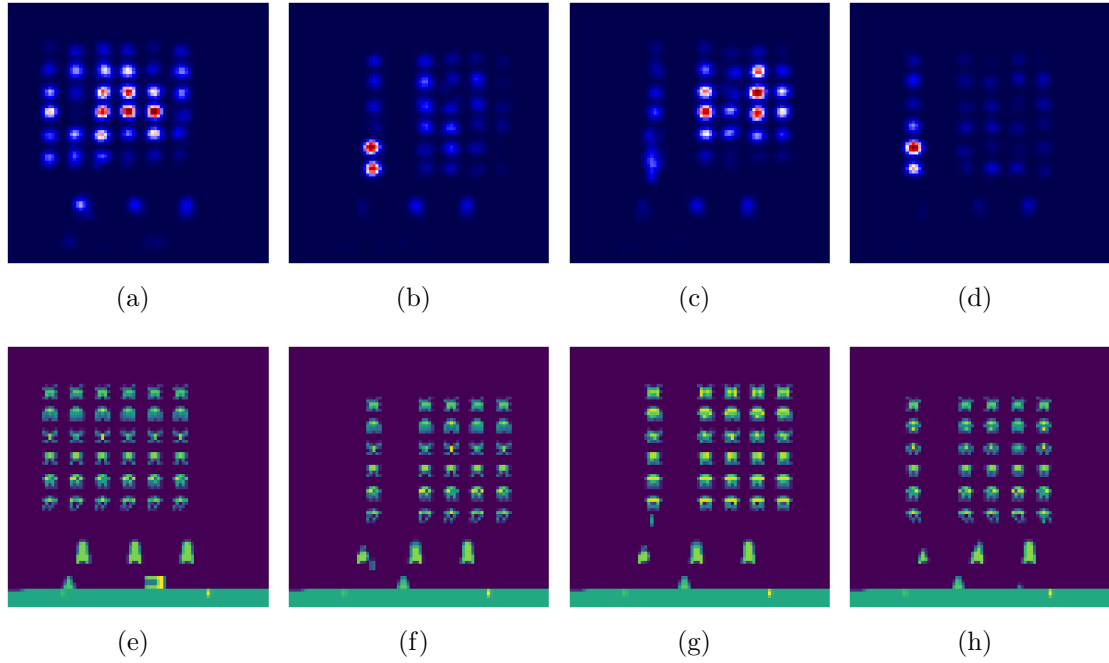


FIGURE 6.2: Column 2 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.2.3 Column 3 Behavior

Figure 6.3 shows the frames and attention maps for the column 3 reward function. We see that in the state of minimum reward, most of the aliens in column 1 are shot, and no aliens in column 3 are shot. This is fine, since it corresponds to an undesirable behavior. In the maximum reward and 95th percentile reward state, 3/6 aliens in column 3 are

shot. The maximum reward frame is not a perfect representation of the optimal state we want (we would desire all of the aliens in column 3 to be eliminated), but these are still desirable states. The 90th percentile reward state has no aliens shot. Based on these observations, we suspect that many of the frames corresponding to our behavior of shooting column 3 are more similar in assigned reward and on the higher end of assigned rewards for this particular reward function than in our other examples. In our other behaviors, the 90th percentile state would be associated with a frame where the desired behavior is being performed.

The attention maps for the high reward states here seem to mostly fixate on aliens remaining in column 3, or nearby aliens to the player that might be shooting at it.

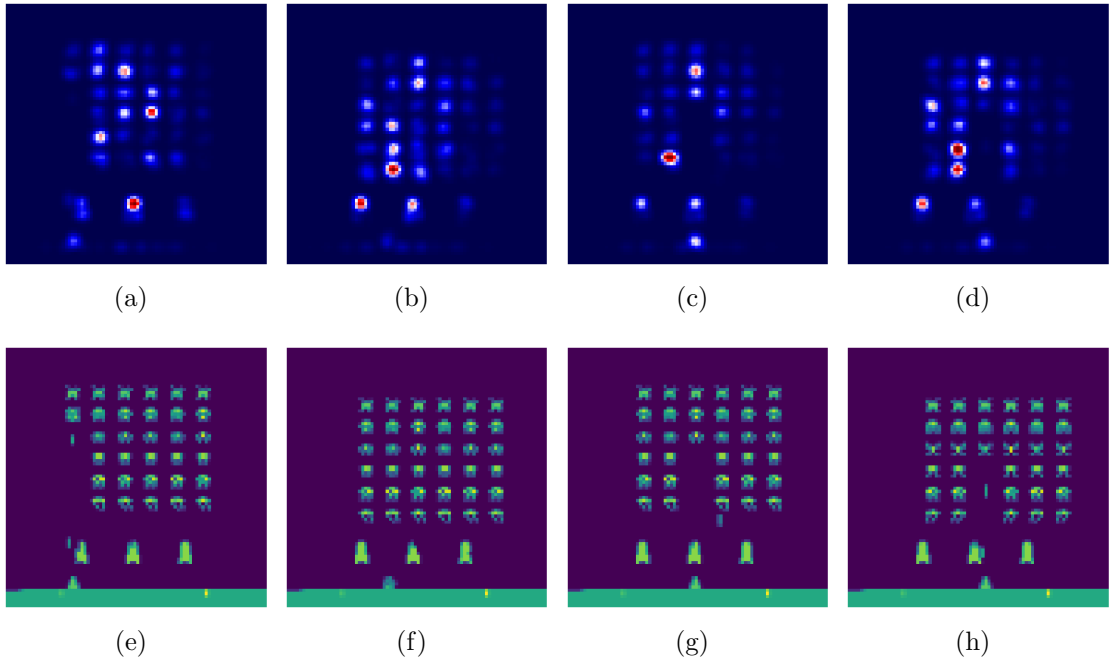


FIGURE 6.3: Column 3 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.2.4 Column 4 Behavior

Figure 6.4 shows the frames and attention maps for the column 4 reward function. We observe that the minimum reward state has no aliens hit, the 90th percentile reward state has 3/6 aliens in column 4 hit and no other aliens hit, and both the 95th percentile and maximum reward states have all aliens in column 4 eliminated. This suggests that the reward function is indeed capturing the desired behavior in high reward states.

The attention map for the 90th percentile state indicates that the reward function pays a lot of attention to the remaining aliens in that column. The other attention maps are less interpretable. One detail that is perhaps interesting is that with the minimum reward attention map, much attention is paid to the 2 rightmost columns instead of column 4, but since the aliens shift position throughout the game, column 4 is *often* located in the area where the pixels are red. We speculate that it is possible that the reward function believes that particular region of the screen is important.

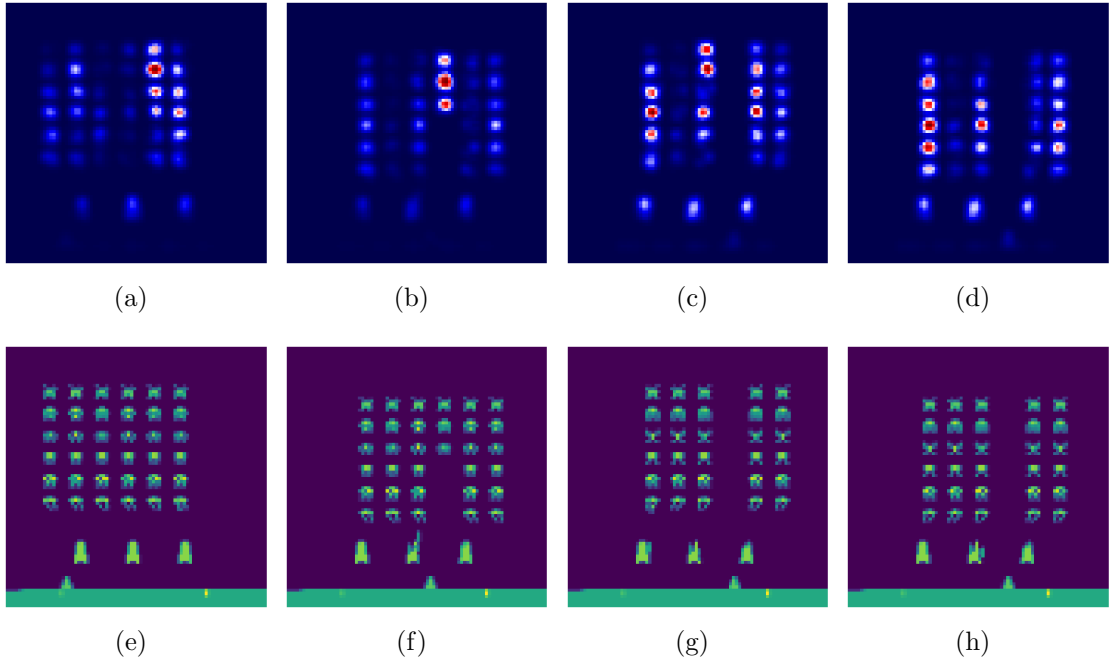


FIGURE 6.4: Column 4 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.2.5 Column 5 Behavior

Figure 6.5 shows the frames and attention maps for the column 5 reward function. We observe that the minimum reward state has no aliens shot, the 90th percentile reward state has 3/6 aliens in column 5 shot, the 95th percentile reward state has 4/6 aliens in column 5 shot, and the maximum reward state has all aliens in column 5 shot. The high reward states are thus being given a gradually more reward as more aliens in the correct column are shot, so the reward function here seems to be representing the desired behavior very well.

The attention maps seem to zone in on the remaining aliens in column 5, as well as some other aliens that are near the column and the player. This seems to support the hypothesis that the reward function cares about aliens that are capable of shooting the player in the near future.

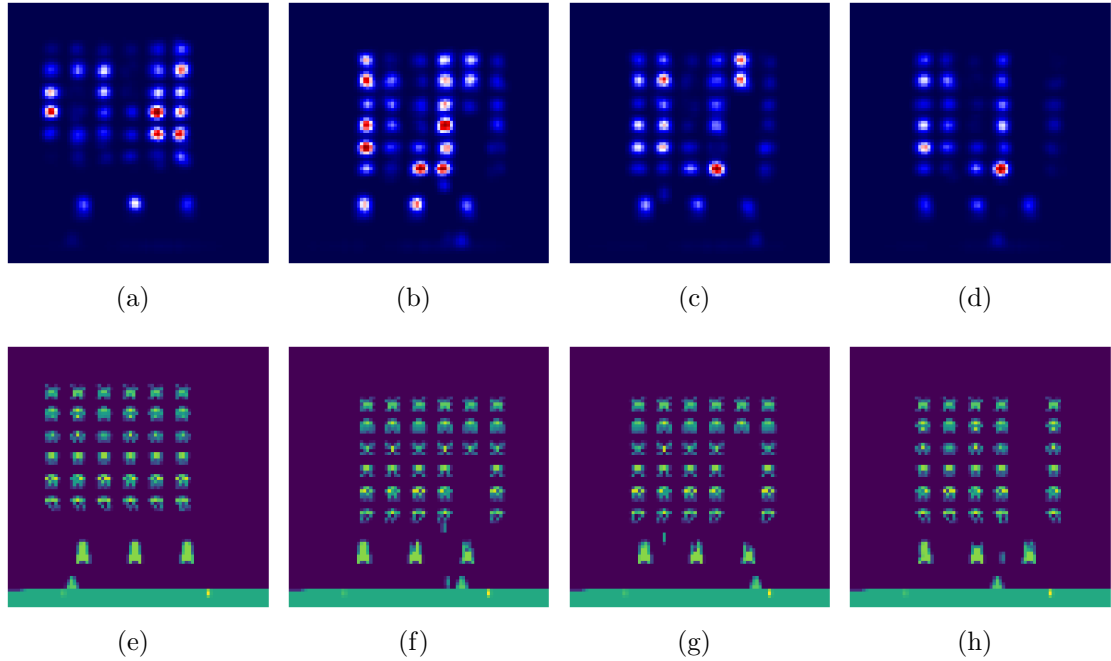


FIGURE 6.5: Column 5 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.2.6 Column 6 Behavior

Figure 6.6 shows the frames and attention maps for the column 6 reward function. The minimum reward state is one in which many aliens have been eliminated, but none are in the correct column. This is fine, since this is an undesirable state. The 90th percentile reward state has 3/6 aliens in column 6 shot, the 95th percentile reward state has 4/6 aliens in column 6 shot, and the maximum reward state has all aliens in column 6 shot (the rest of the aliens move farther to reach the right edge of the screen). This reward function seems to be explaining the desired features of the behavior well.

The attention maps in this example are less interpretable, although the state frames seem to give strong evidence that greater reward is given to states where more aliens in column 6 are eliminated, which is desirable.

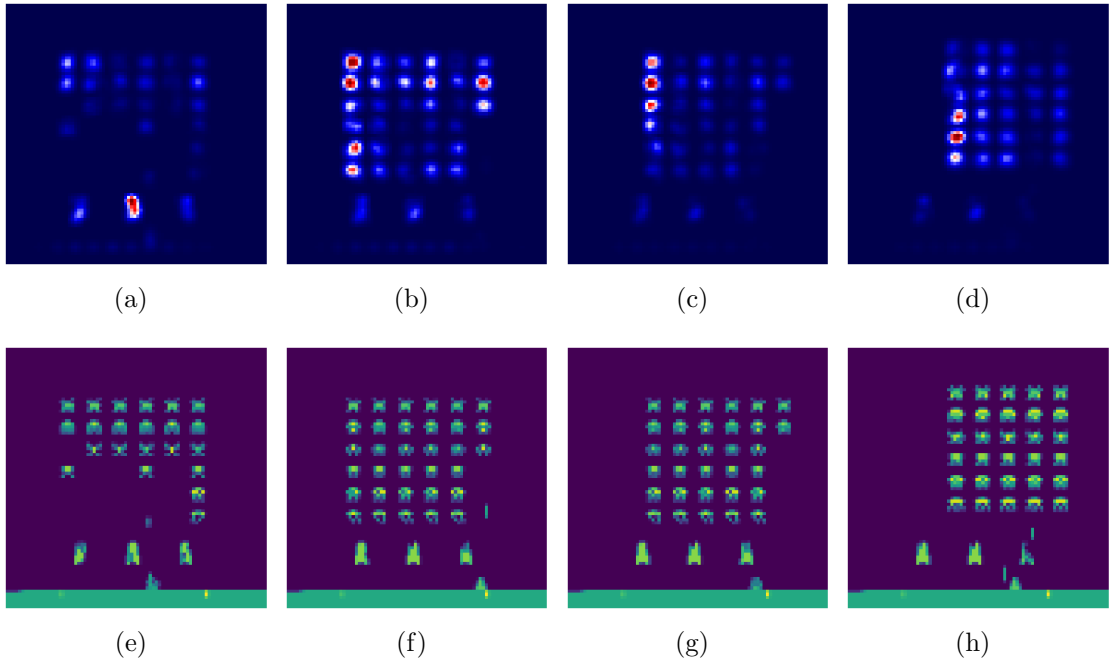


FIGURE 6.6: Column 6 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.2.7 Row 1 Behavior

Figure 6.7 shows the frames and attention maps for the row 1 reward function. We observe that the minimum reward state has no aliens hit. The 90th percentile reward state has 3/6 row 1 aliens eliminated with no other aliens hit, the 95th percentile reward state has 4/6 aliens in row 1 eliminated, and the maximum reward state has all aliens in row 1 eliminated. The high reward states seem to indicate that higher reward is associated with more aliens in the correct row being eliminated, so the reward function seems to capture the desired behavior.

The attention maps appear to focus on aliens that are closer in column position to the player and might pose an immediate threat.

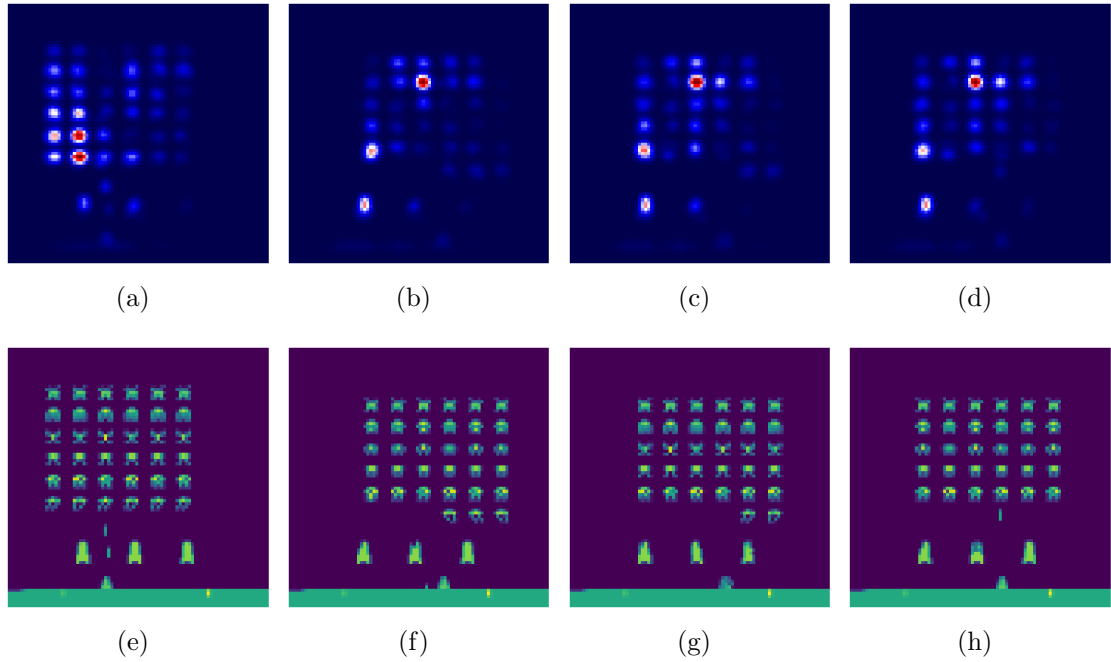


FIGURE 6.7: Row 1 attention maps above and frame images below, for 4 different states. The states shown are assigned reward that is (a/e) minimum; (b/f) at the 90th percentile; (c/g) at the 95th percentile; (d/h) maximum.

6.3 Summary

In this chapter, we describe our setup for investigating reward functions through attention maps and frame states, and then analyze seven learned reward functions. We find that across column and row shooting behaviors, the reward functions are generally assigning higher reward to more desirable states, providing evidence that a ranked demonstration approach can learn reward functions that are good at describing a variety of intentions.

CHAPTER 7

Conclusion

In this thesis, we adapt a ranked demonstration based IRL method to achieve imitation learning results.

In chapter 2, we present standard reinforcement learning algorithms which rely on the environment being able to produce rewards that accurately describe how well an agent is performing. In chapter 3, we present the field of imitation learning, which is motivated by the very common situation in the real world that a reward function is absent, and in applications where we wish for an agent to model human behavior. We ask an agent to mimic an optimal demonstrator, rather than thinking about reward functions.

This problem also motivates inverse reinforcement learning, discussed in chapter 4, where we still attempt to use reward functions, but where we aim to learn them first before using them for standard RL algorithms. In the process of exploring various methods of imitation learning and inverse reinforcement learning, we observe various problems with the standard paradigms in both.

Imitation learning methods often suffer from a lack of understanding of the intent of the expert across large parts of the state space, which is a problem dealt with by IRL through reward learning. Thus, we imagine using reward learning to achieve the goals of imitation learning. However, many IRL approaches are difficult to implement and are computationally expensive, and they often assume optimal expert demonstration quality which may not be available.

The approach of ranked suboptimal trajectories allows us to learn reward functions for behaviors while sidestepping these problems. In chapter 5, we investigate the use of ranked suboptimal human demonstrations to train agents to understand different behaviors in Atari Space Invaders, and propose the algorithm Trajectory Ranked Imitation Learning (TRIL) to imitate expert behaviors. We observe that it is possible to train agents to imitate experts by not only supplying optimal demonstrations of expert behaviors, but bad demonstrations of the desired behavior. We show that this allows us to imitate more complex behaviors than the standard method of behavior cloning allows. Furthermore, in chapter 6 we show that a variety of behaviors can be adequately encoded by reward functions through this method, by investigating high reward and low reward states with respect to these learned reward functions. These results suggest that learning reward functions capturing expert intent is a promising approach to imitation learning.

7.1 Further Work

In the future, we would like to expand this study of trajectory rankings, and more broadly, the learning of reward functions capturing expert intentions, to other environments such as other Atari games, robotics, and driving.

While we largely analyze the quality of imitation in our experiments with the targets hit by imitators in Space Invaders, there are other experiments we could run to evaluate imitation that do not necessarily consider which aliens are hit. We could attempt to imitate player movement instead, for example the behavior of staying on one side of the screen. To further evaluate the quality of imitation, we could look at more comparisons, such as taking a trajectory played out by the imitator and the best demonstration in the training data and seeing if the distribution of actions taken are similar. It is possible that our methods work well as a result of our particular evaluation metrics and the specific behaviors, and more in-depth study of player actions could provide more information about whether this is the case.

In the interest of more rigorous imitation comparisons, it could be worthwhile to study the effects of using different numbers of demonstrations on the learned reward functions, and comparing TRIL with state-of-the-art IL algorithms to evaluate the performance of these methods across different amounts of training data.

We imagine that there are many ways to generate suboptimal trajectories for a reward learner to capture the intention behind a behavior. In order to understand what kinds of suboptimal performances are most helpful for a reward learner, we could conduct a principled study on different qualitative properties of demonstrations submitted to the reward learner for particular behaviors and observe which demonstrations lead to better or worse reward learning.

We observed in a couple of our experiments, such as the columns 4-6 behaviors, that we could learn a reward function that captures intended behaviors, but RL was unable to learn a policy that generates high reward under these reward functions. It would be worthwhile to investigate which kinds of reward functions lead to good results when trained using various RL methods.

We believe it would be fruitful to explore the use of these methods in multi-agent settings as well, where experts adapt their preferences and strategies in response to the behavior of other experts. We could potentially train several agents to each mimic a behavior and put them into the same environment to model multi-agent behavior. In particular, we might imagine submitting ranked preferences over states to train each agent to desire certain objectives over others, and then release the agents into the same environment to see if competitive or cooperative behaviors emerge.

Appendices

A Experiment Setup

In our experiments using trajectory rankings, forward RL is done using PPO (proximal policy optimization) [Schulman et al. 2017] as described in Chapter 2.

For the PPO agents, the learning rate is 3×10^{-4} , the discount factor is $\gamma = 0.99$, and $\epsilon = 0.2$.

In all of our trajectories, we use a technique called frame stacking where we take four game observations and concatenate them into a single observation, which tends to help these algorithms interpret movement over time. We also gray-scale the observed images in Space Invaders and resize them so that each image is of dimension 84×84 pixels. Thus each observation has dimension $(84, 84, 4)$.

For the reward learner, we use a convolutional neural network (CNN) with four convolutional layers and two (linear) fully connected layers. We use a learning rate of 5×10^{-5} . CNNs are described in the next section.

Layer	Input Channels	Output Channels	Kernel Size	Stride
Conv Layer 1	4	16	7	3
Conv Layer 2	16	16	5	2
Conv Layer 3	16	16	3	1
Conv Layer 4	16	16	3	1

Layer	Inputs	Outputs
Fully Connected Layer 1	784	64
Fully Connected 2	64	1

B Convolutional Neural Networks

A convolutional neural network (CNN) is a neural network with at least one convolutional layer that submits inputs into at least one fully connected layer [Krizhevsky et al. 2012]. A CNN often uses the convolutional layers to recognize features in a two dimensional input image by taking connections and corresponding weights attached to the input and using pooling effects locally, which results in the detection of features that are invariant to translations throughout the image, such as edges. In comparison with fully connected networks with a similar number of hidden units, CNNs often have far fewer connections and are therefore much faster to train.

The input to a convolutional layer in our work is an image of dimension $(84 \times 84 \times 4)$, where 84 is the size of the height and width of a frame in Space Invaders, and each observation contains 4 frames. Usually, images come with RGB values encoded into 3 channels, but in our case, we use grayscale values for our images because the color values do not matter for our purposes. The convolutional layer uses *filters* of smaller size in each dimension compared to the input image, and the sizes of these filters determines local structures which are each *convolved* with the input image to produce feature maps of smaller size. Strides determine skip sizes over pixels when applying filters over the image (e.g., a stride of 1 indicates that the filter skips no pixels both horizontally and vertically when being convolved with each section of the input image). In order to reduce the number of parameters of the network, each feature map is then subsampled. The final layers of the CNN are then fully connected layers that reflect the typical architecture of standard neural networks.

C TRIL Demonstrations

For each behavior in our experiments, we generated 12 demonstrations of various quality by playing the game ourselves, and we will qualitatively describe these demonstrations in this section.

For each behavior, we list out and enumerate the demonstration description. An alien is called “correct” if it is in the desired row. The most preferred trajectories are at the top of the list (labeled with a smaller number) and the least preferred trajectories are at the bottom of this list (labeled with a larger number). Only the aliens described are hit in the demonstrations, and no other aliens are targeted.

For more optimal demonstrations, we opt to shoot more of the correct aliens. For the less optimal demonstrations, we opt to shoot more aliens in the incorrect columns or rows in order to discourage states in which incorrect aliens are eliminated.

After submitting these trajectories, we learn a reward function. The reward function then associates total rewards to each of the original demonstrations by summing up rewards across all states reached in each demonstration. The table corresponding to each behavior lists out the total reward given to each of the original demonstrations. Each trajectory number in the table corresponds to the same trajectory number in the list describing the demonstrations.

We observe that in all behaviors, the learned reward functions assign a total reward to each demonstration consistent with their rankings, as desired; that is, if τ_i is preferable to τ_j , then the total reward assigned to τ_i exceeds the total reward assigned to τ_j .

C.1 Column 1

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in column 2 hit.
- (9) Several in column 2 hit.
- (10) Several in columns 2, 3 hit.
- (11) Several in columns 2, 3, 4 hit.
- (12) Much of columns 2, 3, 4 hit.

Associated Rewards

Trajectory	Reward
1	47.9
2	-0.7
3	-139.7
4	-196.3
5	-285.3
6	-366.1
7	-464.5
8	-497.8
9	-551.0
10	-596.1
11	-686.5
12	-774.1

C.2 Column 2

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in column 1 hit.
- (9) Several in column 1 hit.
- (10) Several in columns 1, 3 hit.
- (11) Several in columns 1, 3, 4 hit.
- (12) Much of columns 1, 3, 4 hit.

Associated Rewards

Trajectory	Reward
1	-413.7
2	-661.4
3	-718.5
4	-788.8
5	-893.4
6	-978.0
7	-1027.9
8	-1087.2
9	-1167.3
10	-1214.9
11	-1333.3
12	-1589.2

C.3 Column 3

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in column 2 hit.
- (9) Several in column 2 hit.
- (10) Several in columns 2, 4 hit.
- (11) Several in columns 1, 2, 4 hit.
- (12) Much of columns 1, 2, 4 hit.

Associated Rewards

Trajectory	Reward
1	196.0
2	169.2
3	113.4
4	101.0
5	66.2
6	38.6
7	21.3
8	-9.8
9	-34.7
10	-45.4
11	-81.9
12	-180.9

C.4 Column 4

Although we do not include imitation results for columns 4 through 6 behaviors, we still provide descriptions of the demonstrations that we used to train the reward functions that we looked at in chapter 6.

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in column 3 hit.
- (9) Several in column 3 hit.
- (10) Several in columns 3, 5 hit.
- (11) Several in columns 2, 3, 5 hit.
- (12) Much of columns 2, 3, 5 hit.

Associated Rewards

Trajectory	Reward
1	184.3
2	-66.6
3	-352.3
4	-452.8
5	-531.4
6	-679.1
7	-740.4
8	-792.2
9	-877.3
10	-1059.9
11	-1360.0
12	-1695.9

C.5 Column 5

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in column 4 hit.
- (9) Several in column 4 hit.
- (10) Several in columns 3, 4 hit.
- (11) Several in columns 2, 3, 4 hit.
- (12) Much of columns 2, 3, 4 hit.

Associated Rewards

Trajectory	Reward
1	-749.6
2	-826.1
3	-908.7
4	-1033.2
5	-1095.7
6	-1268.9
7	-1344.0
8	-1405.3
9	-1504.7
10	-1615.0
11	-1775.4
12	-1865.3

C.6 Column 6

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in column 5 hit.
- (9) Several in column 5 hit.
- (10) Several in columns 3, 4 hit.
- (11) Several in columns 2, 3, 4, 5 hit.
- (12) Much of columns 2, 3, 4, 5 hit.

Associated Rewards

Trajectory	Reward
12	-108.6
11	-260.9
10	-348.2
9	-390.6
8	-444.0
7	-480.0
6	-513.6
5	-552.5
4	-605.8
3	-698.2
2	-973.8
1	-1321.2

C.7 Row 1

For the demonstrations of this behavior, we swept the bottom row from left to right so that imitators would be encouraged to move in a particular direction. Thus, for the top 6 demonstrations, we are assuming that the aliens are hit in order from left to right. Some of the suboptimal demonstrations below these involve hitting bottom row aliens out of this order so that this type of movement is discouraged.

Demonstration Descriptions

- (1) 6/6 correct aliens hit.
- (2) 5/6 correct aliens hit.
- (3) 4/6 correct aliens hit.
- (4) 3/6 correct aliens hit.
- (5) 2/6 correct aliens hit.
- (6) 1/6 correct aliens hit.
- (7) No aliens hit.
- (8) 1 alien in middle of row 1.
- (9) Several aliens in middle of row 1.
- (10) Several aliens in and above row 1.
- (11) Even more aliens in and above row 1.
- (12) Many aliens in multiple rows.

Associated Rewards

Trajectory	Reward
1	-467.1
2	-533.5
3	-563.7
4	-591.2
5	-643.2
6	-704.8
7	-790.0
8	-793.3
9	-839.0
10	-871.7
11	-1121.1
12	-1145.6

Bibliography

- Abbeel, Pieter and Andrew Y. Ng (2004). ‘Apprenticeship Learning via Inverse Reinforcement Learning’. In: *Proceedings of the 21st International Conference on Machine Learning*.
- Akkaya, Ilge et al. (2019). ‘Solving rubik’s cube with a robot’s hand’. In: *arXiv preprint arXiv:1910.07113*.
- Arjovsky, Martin, Soumith Chintala and Léon Bottou (2017). ‘Wasserstein GAN’. In: *arXiv preprint arXiv:1701.07875*.
- Bellman, Richard (1954). ‘The theory of dynamic programming’. In: *Technical report, RAND Corporation*.
- Bradley, Ralph Allen and Milton E. Terry (1952). ‘Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons’. In: *Biometrika* 39, pp. 324–345.
- Brantley, Kianté, Wen Sun and Mikael Henaff (2020). ‘Disagreement-regularized imitation learning’. In: *International Conference on Learning Representations*.
- Brown, Daniel S. et al. (2019). ‘Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations’. In: *Proceedings of the 36th International Conference on Machine Learning*.
- Carroll, Micah et al. (2019). ‘On the Utility of Learning about Humans for Human-AI Coordination’. In: *Advances in Neural Information Processing Systems*, pp. 5175–5186.
- Christiano, Paul F. et al. (2017). ‘Deep Reinforcement Learning from Human Preferences’. In: *Advances in Neural Information Processing Systems* 30.
- Codevilla, Felipe et al. (2019). ‘Exploring the limitations of behavior cloning for autonomous driving’. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9329–9338.

- Finn, Chelsea et al. (2016). ‘A Connection Between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models’. In: *arXiv preprint arXiv:1611.03852v3*.
- Fu, Justin, Katie Luo and Sergey Levine (2018). ‘Learning robust rewards with adversarial inverse reinforcement learning’. In: *International Conference on Learning Representations*.
- Goo, Wonjoon and Scott Niekum (2019). ‘One-shot learning of multi-step tasks from observation via activity localization in auxiliary video’. In: *2019 IEEE International Conference on Robotics and Automation (ICRA)*.
- Goodfellow, Ian J. et al. (2014). ‘Generative adversarial nets’. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems 2*, pp. 2672–2680.
- Greydanus, Samuel et al. (2018). ‘Visualizing and Understanding Atari Agents’. In: *International Conference on Machine Learning*, pp. 1787–1796.
- Hessel, Matteo et al. (2018). ‘Rainbow: Combining Improvements in Deep Reinforcement Learning’. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Hester, Todd et al. (2017). ‘Deep Q-learning from Demonstrations’. In: *arXiv preprint arXiv:1704.03732*.
- Ho, Jonathan and Stefano Ermon (2016). ‘Generative Adversarial Imitation Learning’. In: *Proceedings of the 30th Conference on Neural Information Processing Systems*, pp. 4565–4573.
- Hu, Hengyuan et al. (2020). ‘“Other-Play” for Zero-Shot Coordination’. In: *arXiv preprint arXiv:2003.02979*.
- Ibarz, Borja et al. (2018). ‘Reward learning from human preferences and demonstrations in Atari’. In: *Advances in Neural Information Processing Systems 31*.
- Jeon, Wonseok, Seokin Seo and Kee-Eung Kim (2018). ‘A Bayesian Approach to Generative Adversarial Imitation Learning’. In: *A Bayesian Approach to Generative Adversarial Imitation Learning*.
- Kappen, Hilbert J., Vicenç Gómez and Manfred Opper (2009). ‘Optimal control as a graphical model inference problem’. In: *arXiv preprint arXiv:0901.0633*.

- Kingma, Diederik P. and Jimmy Ba (2014). ‘Adam: A Method for Stochastic Optimization’. In: *arXiv preprint arXiv:1412.6980*.
- Kodali, Naveen et al. (2017). ‘On Convergence and Stability of GANs’. In: *arXiv preprint arXiv:1705.07215*.
- Krizhevsky, Alex, Ilya Sutskever and Geoffrey E. Hinton (2012). ‘ImageNet Classification with Deep Convolutional Neural Networks’. In: *Advances in Neural Information Processing Systems 25*.
- LeCun, Yann, Yoshua Bengio and Geoffrey Hinton (2015). ‘Deep learning’. In: *Nature* 521, pp. 436–444. DOI: <https://doi.org/10.1038/nature14539>.
- Liu, YuXuan et al. (2018). ‘Imitation from observation: Learning to imitate behaviors from raw video via context translation’. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1118–1125.
- Luce, R. Duncan (2012). *Individual choice behavior: A theoretical analysis*. Courier Corporation.
- Ma, Yecheng Jason (2020). ‘From Adversarial Imitation Learning to Robust Batch Imitation Learning’. In: Bachelor’s thesis, Harvard College.
- Mnih, Volodymyr et al. (2015). ‘Human-level control through deep reinforcement learning’. In: *Nature* 518, pp. 529–533. DOI: <https://doi.org/10.1038/nature14236>.
- Ng, Andrew Y. and Stuart Russell (2000). ‘Algorithms for Inverse Reinforcement Learning’. In: *Proceedings of the 17th International Conference on Machine Learning*, pp. 663–670.
- Pan, Yunpeng et al. (2017). ‘Agile autonomous driving using end-to-end deep imitation learning’. In: *arXiv preprint arXiv:1709.07174*.
- Pomerleau, Dean A. (1991). ‘Efficient training of artificial neural networks for autonomous navigation’. In: *Neural Computation* 3, pp. 88–97.
- Puterman, Martin L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
- Quinonero-Candela, Joaquin et al. (2008). ‘Dataset Shift in Machine Learning’. In:

- Ratliff, Nathan D., J. Andrew Bagnell and Martin A. Zinkevich (2006). ‘Maximum margin planning’. In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 729–736.
- Reddy, Siddharth, Anca D. Dragan and Sergey Levine (2019). ‘SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards’. In: *arXiv preprint arXiv:1905.11108*.
- Ross, Stéphane and J. Andrew Bagnell (2010). ‘Efficient reductions for imitation learning’. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp. 661–668.
- Ross, Stéphane, Geoffrey J. Gordon and J. Andrew Bagnell (2011). ‘A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning’. In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*.
- Russell, Stuart and Peter Norvig (2020). *Artificial Intelligence: A Modern Approach*. Prentice Hall.
- Sasaki, Fumihiro, Tetsuya Yohira and Atsuo Kawaguchi (2019). ‘Sample Efficient Imitation Learning for Continuous Control’. In: *International Conference on Learning Representations (ICLR)*.
- Schulman, John et al. (2015a). ‘High-Dimensional Continuous Control Using Generalized Advantage Estimation’. In: *arXiv preprint arXiv:1506.02438*.
- Schulman, John et al. (2015b). ‘Trust Region Policy Optimization’. In: *Proceedings of the 31st International Conference on Machine Learning*.
- Schulman, John et al. (2017). ‘Proximal Policy Optimization Algorithms’. In: *arXiv preprint arXiv:1707.06347v2*.
- Sermanet, Pierre et al. (2018). ‘Time-contrastive networks: Self supervised learning from video’. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1134–1141.
- Silver, David et al. (2016). ‘Mastering the game of Go with deep neural networks and tree search’. In: *Nature* 529, pp. 484–489. DOI: <https://doi.org/10.1038/nature16961>.

- Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. MIT Press.
- Sutton, Richard S. et al. (1999). ‘Policy gradient methods for reinforcement learning with function approximation’. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pp. 1057–1063.
- Torabi, Faraz, Garrett Warnell and Peter Stone (2018). ‘Behavioral cloning from observation’. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*.
- Watkins, Chris and Peter Dayan (1992). ‘Q-learning’. In: *Machine learning* 8, pp. 279–292.
- Williams, Ronald J. (1992). ‘Simple statistical gradient-following algorithms for connectionist reinforcement learning’. In: *Machine Learning* 8, pp. 229–256.
- Wulfmeier, Markus, Peter Ondruska and Ingmar Posner (2015). ‘Maximum Entropy Deep Inverse Reinforcement Learning’. In: *arXiv preprint arXiv:1507.04888*.
- Yu, Tianhe et al. (2018). ‘One-shot imitation from observing humans via domain-adaptive meta-learning’. In: *arXiv preprint arXiv:1802.01557*.
- Zhang, Jiakai and Kyunghyun Cho (2016). ‘Query-efficient imitation learning for end-to-end autonomous driving’. In: *arXiv preprint arXiv:1605.06450*.
- Ziebart, Brian D. et al. (2008). ‘Maximum Entropy Inverse Reinforcement Learning’. In: *Association for the Advancement of Artificial Intelligence*.