Modeling Human Behavior in Space Invaders

Modeling Human Behavior in Space Invaders

A thesis presented

by

James Lennon

to

The Department of Computer Science

in partial fulfillment of the requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

March 2019

Thesis Advisor: Professor David Parkes James Lennon

Modeling Human Behavior in Space Invaders

Abstract

Effective AI systems in the real world must be able to interact and cooperate effectively with the people who use and benefit from them. In order to make this possible, these systems must have a realistic model of how humans will behave in various situations; either overestimating or underestimating human performance can lead to strongly suboptimal outcomes. To this end, this thesis proposes a new algorithm for imitation learning, working in the Atari 2600 Space Invaders environment. We first modify GAIL, a state-of-the-art deep imitation learning algorithm, to work in Atari environments and verify that it scales up to more complex environments more effectively than the original version of the algorithm. We then build a framework for evaluating and comparing human imitators, developing a set of relevant statistics that consider both in-environment performance and descriptive similarity. The new method that is introduced breaks down the problem of human imitation into two subproblems: creating an agent that plays the game well and learning a "corrective" function that modifies this agent to play in a human manner. This hybrid approach is fast to train and can be easily tuned along a spectrum to make the tradeoff between more closely matching the human behavior or performing at a higher level. This approach shows promising results across the evaluation statistics; it achieves a high likelihood of the data under the learned policy, produces a score distribution matching that of the human data, and also matches the human distribution of actions as it acts in the environment.

Contents

List of Figures

Acknowledgments

I am indebted to Professor David Parkes for his guidance, encouragement, and mentorship throughout the process of this thesis. I would like to thank Paul Tylkin and Goran Radanovic for their constant inspiration, encouragement, and generosity of their time working with me on this project.

I thank Professors Finale Doshi-Velez and Sasha Rush for generously offering to be my thesis readers.

I would also like to thank Alan Estrada, Charles Liu, Janet Chen, Jon Suh, Liam Hackett, Majo Acosta, Marcella Park, Mattia Mahmoud, Nick Pham, Ritayan Chakraborty, Sierra Nota, Tom Orton, and Vladislav Sevostianov for kindly providing data of human play in Space Invaders that was essential for the development of this thesis.

I would last like to thank my mother and father for encouraging and supporting with this project.

Chapter 1

Introduction

AI systems are becoming increasingly prevalent and have the potential to revolutionize many aspects of daily life from transportation to grocery shopping. In order to be effective, these systems must be able to successfully interact with the people who benefit from and cooperate with them.

1.1 Human-AI Cooperation

To enable this kind of beneficial cooperation between humans and AI systems, we must train our algorithms to properly understand how people will behave in various scenarios, neither overestimating nor underestimating human performance. Consider self-driving cars; as these AI systems become increasingly adopted, they must be able to effectively cooperate with human drivers if they are to succeed. Assuming that other drivers will perform at the same level as a self-driving car is potentially detrimental, as the AI may not take into account situations in which humans are prone to accidents. On the other hand, assuming that other drivers behave at a sub-human level will lead to poor performance and overly cautious driving. Additionally, assistive AI technologies have huge potential to help the elderly or people with disabilities, but it is critical that these technologies are able to effectively cooperate with their users. Thus, it is essential for these AI systems to have an accurate model of human behavior so that they can be trained to maximize performance in cooperative environments with humans.

Figure 1.1 shows a general framework for generating a Helper-AI using crowdsourced behavior data, a human imitator, and two-player reinforcement learning [11]. The process first starts with an AI agent that has been trained to

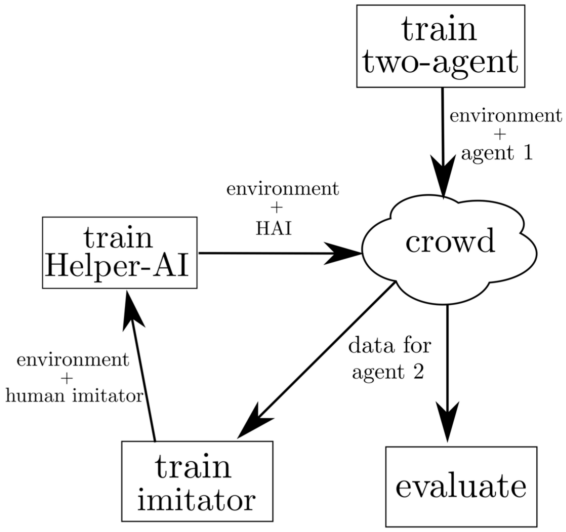

Figure 1.1: Helper-AI Framework (image from [11])

assume that the other player will behave the same as itself ("train two-agent" in the diagram). The framework then collects data from a "crowd" of humans interacting in the environment with this agent, and thens train a human imitator to replicate the human behavior when cooperating with the AI. We then train the Helper-AI again, except this time we use the human imitator to play alongside the AI during training so that it has a sense of how to best cooperate with the human ("train Helper-AI" in the diagram). Once we have this new agent, we must collect more data of how humans will cooperate with the AI since the its behavior is different and humans may react differently. We repeat this cycle until performance converges and we reach a fixpoint of this process. This thesis focuses on the "train imitator" portion of this procedure.

1.2 Accounting for Human Behavior

Existing theoretical work demonstrates that there are significant gains in joint score by having the stronger agent incorporate knowledge about a weaker partner's skill level [3]. This claim is further confirmed by research showing that in two-player, cooperative Space Invaders, a Helper-AI can improve joint performance by 49% compared to a weak agent cooperating with another copy of itself [11]. Additionally, pairing a weaker agent with a "strong" AI that doesn't take into account the sophistication level of its partner can lead to 17% worse performance than if the weaker agent was paired with another copy of itself. While the referenced research

used weak AI agents and not humans, we expect to experience similar effects with real humans.

One way to build a Helper-AI that has a model of its partner's behavior is to train an AI system "online," where a human is participating in the designated task with an AI during training. While this would allow the algorithm to have an accurate understanding of human behavior, such an approach would require far too much training time, driving up costs and drastically slowing the development and deployment of new systems. Additionally, there are ethical concerns of training an AI system alongside humans where adverse performance can harm the human, such as training a self-driving car alongside real drivers.

A better approach would be to simulate the task and its environment so that we no longer have to slow down the training process to accommodate human response times. To do this, we need an algorithmic proxy for the human that is able to perform the task in a human-like way, capturing the idiosyncrasies of human behavior. To do this, the human model must learn from a dataset of human behavior in order to properly replicate it. This thesis will show that it is possible to accurately model and simulate human behavior in a complex environment, investigating several methods for evaluating the performance of various proxy algorithms to measure the "human" quality of their performance.

The goal of this project is to build a generalizable algorithm for modeling human behavior. Given a dataset of human behavior, it should be able to replicate this behavior so that it is indistinguishable from the human. Note that the objective is not to maximize for performance, but rather to maximize for similarity to human behavior. This algorithm allows us to train AI agents alongside a human model to maximize the joint score between the human and the AI, thus optimizing for cooperation.

1.3 Space Invaders

The real world is messy and not easily interpreted; effective AI systems may require the use of sensor networks or other high-dimensional data to infer the state of the world. For this reason, the OpenAI Gym [1] Space Invaders environment is a good choice for building this algorithm to model human behavior. In this environment, the task is to learn to maximize score using only the pixel values of the screen as input. Similar to how a self-driving car must learn to identify other cars from LIDAR and visual data, this system must learn to visually decode the state of the

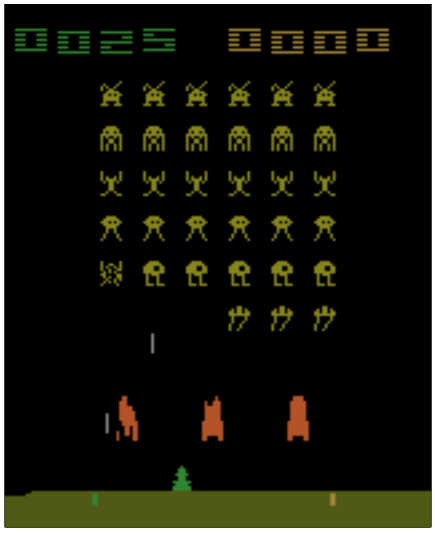

Figure 1.2: Space Invaders

environment from the high-dimensional pixel values of the game. Past work shows that it is possible to achieve super-human performance in Atari environments from raw pixel values and scores [10]. For these reasons, we choose Space Invaders as a testing ground for human imitation because this environment presents unique challenges; success in this domain will demonstrate that it is possible to accurately imitate human behavior in similarly complex tasks.

In Figure 1.2, we see an example frame from Space Invaders. The player controls the ship at the bottom of the frame, and the ship can move horizontally left and right. The objective is to shoot all of the yellow aliens as they slowly descend to the ground. The aliens also shoot bullets at the player; if the player hits a bullet they lose a life, and once they lose three lives the game ends. There are three orange shields above the ship that the player can hide below for cover, but these can be damaged by bullets. The action set includes moving right and left, firing, moving left and right while firing, and a no-op action.

1.4 Limitations of Existing work

The goal of learning to replicate human behavior falls under the umbrella of *imitation learning*, where an agent learns to perform a task from example data of some other agent's behavior. Past work on this topic is able to use human data to optimize for score [5; 16]; however, these algorithms are not designed for complex environments where an agent must learn directly from pixel values. Additionally, these algorithms

are aimed at producing optimal behavior and are not designed to capture the flaws inherent in human behavior. Chapter 2 gives background information and discusses imitation learning in more detail.

Recent research in the fields of deep reinforcement learning and imitation learning has focused on Atari video game environments. As discussed earlier, these video games are challenging learning environments and have been used as excellent testing grounds for advanced RL algorithms. In order to model human behavior in complex environments, this thesis builds on past work by combining state-of-the-art imitation learning algorithms with effective deep reinforcement learning algorithms for Atari environments.

1.5 Overview of Contributions

The main contributions of this thesis are summarized below:

- Modifications are outlined for GAIL, a state-of-the-art deep imitation learning algorithm, to work in Atari environments. These improvements are tested it to verify that it scales up to complex environments and policies more effectively than the original version of the algorithm.

- A framework is then described for evaluating and comparing human imitators based off of a set of relevant statistics. A computational procedure for computing these comparisons in an efficient and flexible way is provided.

- The first approach to imitating human behavior is an additional modification to GAIL that incorporates environment reward into GAIL's reward function. This approach is implemented and evaluated, showing that while it does improve the log-likelihood of the data with respect to GAIL, it still does not perform at a similar level as the human.

- The best-performing contribution of this thesis is a novel imitation learning algorithm that combines reward-maximizing behavior and knowledge of human behavior from the dataset. This approach is a better imitator for the human data than GAIL and our previous approaches as evidenced by nearly all of our comparison metrics.

1.6 Outline of Thesis

This thesis is structured as follows:

- Chapter 2 gives an overview of reinforcement learning (RL) and relevant deep RL and imitation learning algorithms. Later chapters will refer to and build off of these algorithms throughout this thesis.
- Chapter 3 discusses modifying a state-of-the-art deep imitation learning algorithm GAIL to work in Atari environments. This modified algorithm is tested on simple policies to make sure that it scales up properly to more complex behavior.
- Chapter 4 describes a framework for evaluating and comparing human imitators based off of a set of relevant statistics. It also provides an overview of how these comparisons are computed in an efficient and flexible way.
- Chapter 5 addresses the limitations of previous approaches and describes a novel approach to imitation human behavior using agents that are hybrids of reward-maximizing behavior and human imitation (Hybrid Agents). The comparison framework from the previous chapter is used to measure the performance of these agents and verify that they provide a faithful representation of human behavior.
- Chapter 6 concludes.

Chapter 2

Background on Deep Reinforcement Learning and Imitation Learning

This chapter provides background and notation for reinforcement learning that will be used throughout this thesis.

2.1 Markov Decision Processes

In reinforcement learning, the environment is typically modeled as a Markov Decision Process (MDP). An MDP represents the state of the environment at each time t as $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the state space. At time $t = 0$, the starting state of the MDP is defined by ρ, which is a distribution over $\mathcal{S}$. At each timestep, an agent selects an action $a_t \in \mathcal{A}$, where $\mathcal{A}$ denotes the action space. When an agent takes action a_t from state s_t, the distribution of its next state s_{t+1} is $P(s_{t+1}|s_t, a_t)$, where $P(\cdot|s, a)$ is the transition model of the MDP. After taking this action, the agent experiences a reward $r_t = R(s_t, a_t, s_{t+1})$, where $R(s, a, s')$ is the reward model. Thus the MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \rho, P, R \rangle$.

An agent acts in the MDP according by taking action $a_t \sim \pi(\cdot|s_t)$, where $\pi(\cdot|s)$ is the agent's policy function. Let $0 < \gamma \leq 1$ be the discount rate of future rewards. The agent optimizes performance in the MDP by maximizing total discounted reward, which for a trajectory of T timesteps is $r = E\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right]$, where starting state $s_0 \sim \rho$, actions are chosen according to the policy function $\pi_\theta(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$.

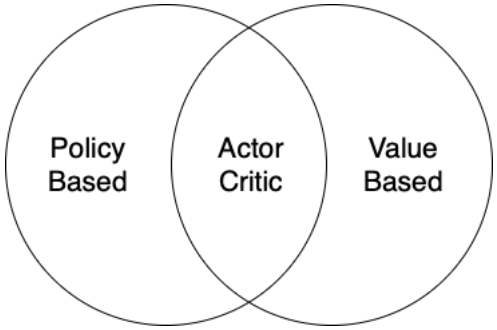

Figure 2.1: Deep RL Categories

2.2 Deep RL Algorithms

The purpose of reinforcement learning (RL) is to optimize performance in an MDP environment, in other words, to maximize total discounted expected reward. The agent must learn to do this while interacting in the environment, without any prior knowledge of the reward structure or the transition model. Recent work has shown that using deep neural networks to implement the agent's policy can be quite successful in maximizing performance in complex environments [10; 13; 17]. By using a deep neural network to implement the policy, the scalability of the algorithm is improved, and it is able to learn more complex behaviors.

Three common categories of deep RL algorithms are value-based, policy-based, and actor-critic (Figure 2.1). Value-based methods focus on learning the value function of the MDP such that they can accurately predict the value of a state and act accordingly. Policy-based methods work to directly optimize the agent's policy to maximize total reward. Actor-critic algorithms are both value-based and policy-based, as they contain two components, an actor which learns the policy, and a critic which learns to predict the value of a state. This is discussed in more detail below in a summary of some of the relevant deep RL algorithms that are referred to in this thesis.

2.2.1 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) is a reinforcement learning method that optimizes the policy such that at each iteration, there is an upper bound on the amount by which the new policy differs from the previous policy [13]. The policy $\pi_\theta(a_t|s_t)$ is represented by a neural network with weight vector θ that takes an input vector representing the state and produces a distribution over actions. The

algorithm then runs in the environment and samples states, actions, and reward values using the policy.

Let $\eta(\theta)$ be the total expected discounted reward for the policy with parameter vector θ, and our goal is to maximize η. Let $\rho_\theta(s)$ be the discounted visitation frequency of state s, so that $\rho_\theta(s) = \sum_{t=1}^{\infty} \gamma^{t-1} P[s_t = s]$. Let $A_\theta(s,a) = Q_\theta(s,a) - V_\theta(s)$ be the advantage function for policy π_θ, which measures the relative benefit of taking action a from state s (discussed further in Section 2.2.2). Using these definitions, we define a local approximation to $\eta(\theta_0)$:

$$L_\theta(\theta') = \eta(\theta) + \sum_s \rho_\theta(s) \sum_a \pi_{\theta'}(a|s) A_\theta(s,a) \tag{2.1}$$

Since this is a local approximation, we know that $\nabla L_\theta(\theta) = \nabla \eta_\theta(\theta)$, so if the difference between θ and θ' is small then updating $\theta \to \theta'$ will increase η as well as L_θ. We ensure that the step size is small by enforcing a constraint on the Kullback-Leibler Divergence (KL-Divergence) between the two policies, $\pi_\theta(a|s)$ and $\pi_{\theta'}(a|s)$. We measure this divergence by taking the mean KL-Divergence at each state, weighted by the visitation frequencies ρ: $\bar{D}^\rho_{KL}(\theta, \theta') = E_{s\sim\rho}[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta'}(\cdot|s))]$.

To maximize this quantity, we first simulate our policy $\pi_{\theta_{\text{old}}}$ in the environment to sample $\rho_{\theta_{\text{old}}}$ and $A_{\theta_{\text{old}}}$. We use these samples to run our maximization step, where δ is the constraint on the KL-Divergence at each step:

$$\theta = \underset{\theta}{\text{argmax}}\, L_{\theta_{\text{old}}} \text{ such that } \bar{D}^{\rho_{\theta_{\text{old}}}}_{KL}(\theta_{\text{old}}||\theta) \leq \delta \tag{2.2}$$

The limit on the amount that it changes the policy at each iteration (the "trust region") has theoretical gains in stability and training time, and this algorithm has been empirically demonstrated to be effective in complex continuous control environments.

The network architecture used in the original TRPO paper for continuous control environments is shown in Figure 2.2. It consists of two fully-connected layers with rectified linear unit (ReLU) activations between each layer.

2.2.2 Advantage Actor-Critic

Actor-critic methods for reinforcement learning have demonstrated excellent generalizability and high performance in Atari environments [9]. These methods involve a value function $V_\theta(s_t)$ as well as the policy function $\pi_\theta(\cdot|s_t)$, parameterized by weights θ [14; 2]. The value function learns to approximate the future expected discounted reward from s_t (the "critic"), and the policy learns how to act at state

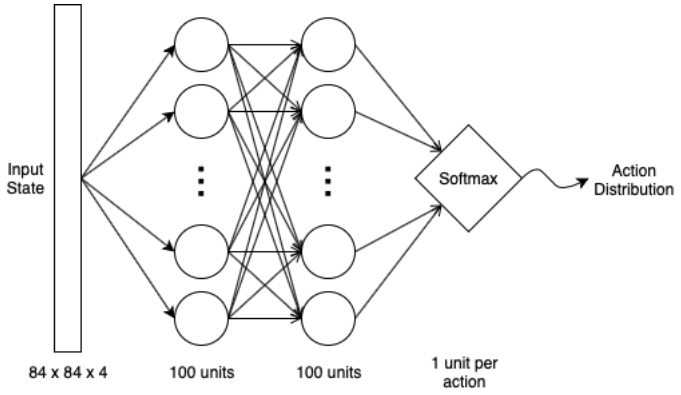

Figure 2.2: TRPO Neural Network Architecture

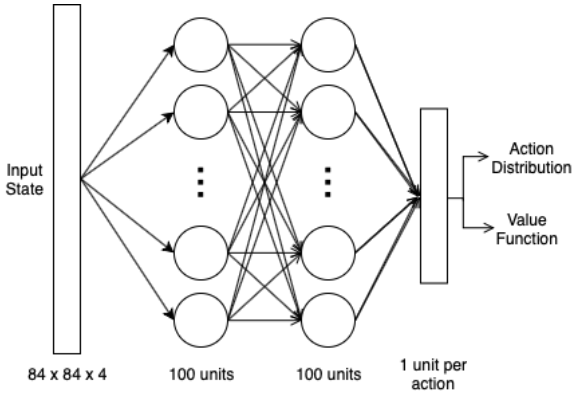

Figure 2.3: A2C Neural Network Architecture

s_t (the "actor"). The agent acts in the environment for T timesteps, recording states $\{s_t\}_{0\leq t\leq T}$, actions $\{a_t\}_{0\leq t<T}$, and rewards $\{r_t\}_{0\leq t<T}$. It then computes $R_t = \sum_{i=t}^{T-1} r_i * \delta^{i-t} + V(s_T)$, where the last term uses the value function to estimate the remaining rewards from the final state of the sample.

Let $L_V = \frac{1}{T}\sum_{t=0}^{T-1}(R_t - V(s_t))^2$ be the loss function for the critic; the algorithm minimizes L_V with respect to θ using gradient-based methods. Let $\hat{A}(s_t) = R_t - V(s_t)$ be the approximate advantage function which represents the additional gain or loss by taking action a_t from state s_t. This function computes the difference between the experienced reward which is the R_t term, and the predicted value of state s_t which is the $V(s_t)$ term. Note that this is an empirical form of the true advantage function $A(s,a) = Q(s,a) - V(s)$.

To encourage exploration, we want to maximize the entropy of the action distributions produced by the actor. Let $H(X)$ be the entropy of distribution X and let β be the coefficient of entropy maximization. The loss function for the actor is then $L_\pi = \frac{1}{T}\sum_{t=0}^{T-1}\Big[\ln(\pi(a_t|s_t))\hat{A}(s_t) - H(\pi(a_t|s_t))\Big]$; the algorithm minimizes this

loss at each iteration. This algorithm is known as Advantage Actor-Critic (A2C).

Figure 2.3 shows the neural network architecture for the model representing the neural network policy π_θ. We use a similar architecture as in Figure 2.2, with the addition of an output of the model that gives the predicted value for the input state, $V_\theta(s)$.

2.3 Imitation Learning Algorithms

The goal of imitation learning is to leverage behavioral data of an existing agent (human or artificial) to learn how to perform in the environment. Deep learning allows for sophisticated imitation learning algorithms; two algorithms relevant to this thesis are summarized below. Note that these algorithms do not take rewards from the environment into account.

2.3.1 Behavioral Cloning

Behavioral cloning treats imitation learning as a supervised learning problem, training a model to predict the action distribution of an agent given a state s_t. The loss for the policy $\pi_\theta(\cdot|s)$ for a batch M state-action pairs is the mean log likelihood of the learned policy, so $L_\pi = \frac{1}{M}\sum_{t=1}^{M} \ln(\pi(a_t|s_t))$. In each batch, a_t and s_t are sampled from the data of the recorded agent.

2.3.2 GAIL

Recent work has shown that generative adversarial AI algorithms are effective at generating highly realistic images that imitate a dataset [4]. These Generative Adversarial Networks (GANs) have become increasingly successful at generating images as well as samples of other complicated distributions. In a similar vein, Generative Adversarial Imitation Learning (GAIL) is a state-of-the-art imitation learning algorithm that uses an adversarial approach [5]. It works by simultaneously training two components: the discriminator and the generator; an overview of this architecture is shown in Figure 2.4.

The discriminator is a binary classifier whose objective is to distinguish between real behavior from the human dataset and fake behavior produced by the generator. Specifically, the discriminator takes as input a single "transition" (s_t, a_t) and

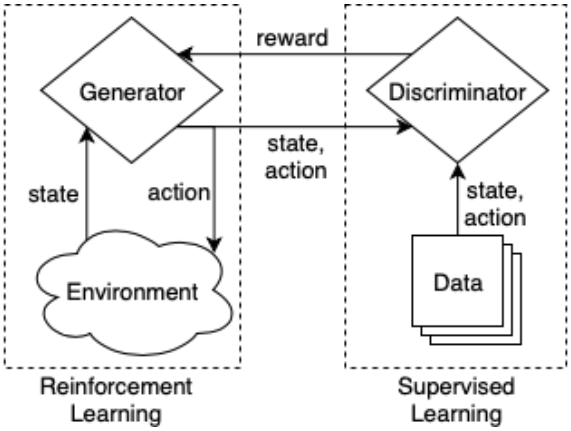

Figure 2.4: GAN Approach for Imitation Learning

classifies it by giving a probability that the transition is human, a value between 0 and 1. Let $D : \mathcal{S} \times \mathcal{A} \to [0, 1]$ be the discriminator function, where an output value of 0 signifies that the transition was generated and 1 signifies that it was produced by the human.

The generator acts as the policy function π_θ and produces trajectories, and its objective is to maximize the cumulative loss function of the discriminator over k steps. Doing so maximizes the amount by which the discriminator is "fooled" into thinking that the transition was from the human dataset. This is achieved by setting the reward function of the MDP in which we train the generator to be $R(s_t, a_t) = -\ln(1 - D(s_t, a_t) + \alpha)$, where α is a small constant to prevent computing the log of 0. This results in the agent experiencing high reward when the discriminator outputs a value close to zero (indicating that it believes the transition was from the human) and vice versa. We train θ to maximize the total discounted reward in this MDP while simultaneously training D to minimize the discriminator loss. This process results in a discriminator that is increasingly accurate and a generator that behaves increasingly similarly to the behavior data. In each iteration of GAIL, we train the generator for g training steps and the discriminator for d training steps. In practice, we set $g > d$ since the generator has a more complex task to learn.

Chapter 3

Modifying and Evaluating GAIL in Atari Environments

Space Invaders and other Atari environments have high-dimensional state spaces and discrete controls, which present challenges when imitating human behavior. This chapter discusses the limitations of GAIL in Atari environments and the necessary modifications to the algorithm to overcome these difficulties. This algorithm is tested on a set of simple hard-coded policies, termed "baseline policies". To design an algorithm capable of replicating arbitrary human behavior in a complex environment, this algorithm must be capable of replicating simpler components of complex behavior. To this end, testing the algorithm on baseline policies is useful for investigating and measuring the performance of the modified GAIL algorithm when imitating behavior.

3.1 Modifications to GAIL

Generative Adversarial Imitation Learning (GAIL) is a state-of-the art imitation learning algorithm that has demonstrated success in continuous control environments [5]. However, improvements are necessary for this algorithm to work in the discrete, high-dimensional Space Invaders environment and allow it to visually process the state from pixel values.

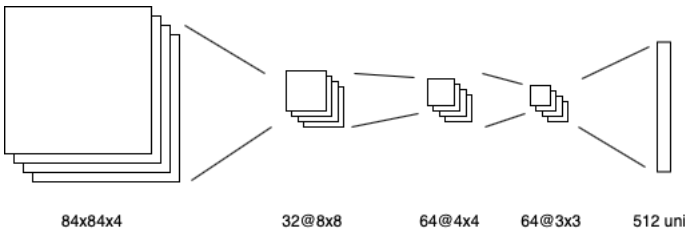

Figure 3.1: Deep Q-Network Convolutional NN Architecture

3.1.1 Network Architecture

To do this, the neural networks used for both the generator and discriminator in GAIL are modified by adding convolutional layers before two fully connected layers. The same convolutional layer structure as in the Deep Q-Network (DQN) algorithm is chosen, since this architecture has demonstrated success in Atari environments including Space Invaders [10]. This architecture is shown in Figure 3.1. The network has 3 convolutional layers with Rectified Linear Unit (ReLU) activations followed by a fully-connected layer that yields a vector of 512 units. For the generator, this vector is then fed through the two-layer fully connected network shown in Figure 2.2.

The discriminator receives an action as input in addition to the state vector; this is represented as a one-hot vector and is concatenated to the 512 output units from the CNN. The resulting vector is passed through the same fully-connected network shown in Figure 2.2. After the fully connected layers the network results in a binary classifier, implemented as a single logit passed through a sigmoid function.

3.1.2 A2C vs TRPO

In the original GAIL paper, TRPO is used to train the generator agent [5]. As discussed earlier, TRPO is successful in continuous control environments, such as robotic control or locomotion. However, Space Invaders has a discrete state space of pixel values, which presents different challenges than these continuous environments. A fundamental difference between these environments is that in continuous control, the action space is a continuous vector of real numbers, whereas in Atari environments, the actions are selected from a discrete set of options. Additionally, the state space in Atari is represented as a grid of pixel values that are selected from a discrete set of possible values instead of a vector of floating-point features.

TRPO makes small controlled adjustments to the policy at each iteration

to smoothly make adjustments along the policy gradient. This makes sense in a continuous environment since the gradient will likely be smooth with respect to the actions, and making small adjustments will likely lead to good performance. However, this is less suitable for Atari environments because of the differences listed above (i.e. pressing a button "harder" doesn't affect performance; it must change the action that is selected). For these reasons, the improved implementation of GAIL uses Advantage Actor-Critic to train the generator instead. As discussed earlier, A2C is effective for maximizing performance in Atari environments, and this chapter will show that it more effectively imitates behavior in Space Invaders from a dataset as well.

3.2 Baseline Policies

These baseline policies are implemented using finite automaton-like implementations of simple behaviors, where the action at each timestep only depends on t. Note that GAIL only receives state-action pairs and not entire trajectories, so this may seem counterintuitive that it is able to learn a policy that is time-dependent. However, it is still possible to learn these policies by using contextual information such as the position of the ship - this is useful because testing GAIL's performance on these policies will force it to learn to parse the relevant contextual information.

The first time-dependent baseline is a policy where the agent moves right until it is directly below the middle shield, at which point it remains there without shooting indefinitely. The hard-coded policy is implemented by simply counting the number of steps to get to the middle shield, and every time the player loses a life, it repeats this procedure. This is an interesting task to replicate because it requires the algorithm's visual system to recognize the middle shield and use this contextual information to inform its action selection. The last and most complex baseline policy is to move to the central shield, and move left a few times and shoot just out of the shield, and then move right a few times to return to the shield, repeating this process indefinitely. This again requires the algorithm's visual system to identify the shield's position and width. It also needs to determine if the ship is moving right or left to make sure that it replicates the smooth left and right movement.

Realistic human behavior can be thought of as a sequence of these basic units of behavior. Demonstrating that our algorithm is capable of replicating these basic behaviors will provide evidence that this algorithm is able to imitate more complex human play.

3.3 Adding Dataset Noise

A common problem that arises with training GANs is generator collapse: when the discriminator becomes too accurate and the generator can no longer fool it, resulting in small and useless gradients for training the generator [8; 12]. A similar issue is experienced with GAIL, as it has a similar adversarial structure where the generator's task is more difficult to learn than the discriminator's task.

These baseline policies present an especially challenging problem for the generator: If the generator makes a single mistake along its trajectory, it gets to a state space that isn't present in the data. This means that the discriminator can easily identify the agent as not human, and the generator has collapsed for the duration of the trajectory. This is a property of the unrealistically "clean" baseline data and is not true of real human performance. To solve this problem we add noise to the data to make it better resemble human behavior and to make the discriminator's job harder. This noise is added by having the policy in the data choose a random action with probability ϵ instead of always taking the next action in the policy. This has the effect of making the discriminator less strict, as a mistake on the part of the generator does not necessarily prove that it isn't from the dataset since there are noise actions in the data.

3.4 Results

The first experiment in this chapter investigates the performance of GAIL when learning the baseline policy that moves to the central shield and stops there; this policy is named "move_to_shield". Figure 3.2 compares the performance when using A2C or TRPO as the generator algorithm and also demonstrates the effects of varying the amount of noise in the data ϵ. This figure measures the policy log-likelihood of the algorithm during training; this metric (described in more detail in Section 4.2.6) measures the predictive accuracy of the model, in other words how likely the learned policy is to replicate the behavior in the data. On the left, we see that TRPO struggles to learn much at all; it seems that in this environment, the algorithm's imitation of the human doesn't improve with training time. It seems to only stay around a constant level and get slightly worse over time. Even though we added convolutional layers to the agent's model in this algorithm, it is still unable to learn to perform well in this environment. Adding noise in the dataset seems to only make performance worse, as we see that the trials with higher ϵ values have lower policy log-likelihoods.

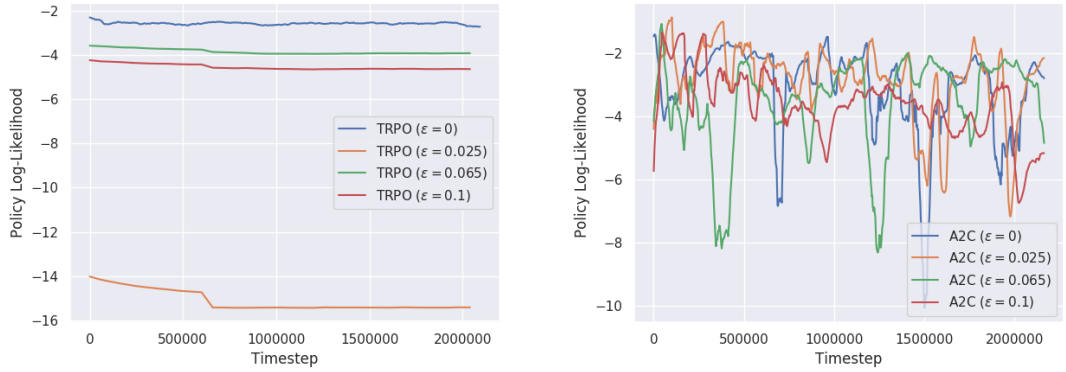

Figure 3.2: Learning move_to_shield with TRPO and A2C

In the next policy, the agent moves to the central shield and then repeats moving left, shooting, and moving right ("shield_shoot"). The results of this experiment are shown in Figure 3.3 on the left. We see the same trend when training GAIL with TRPO on the shield_shoot policy. As explained earlier, TRPO has a smooth learning curve, but it isn't able to make the right changes to accurately imitate the data. It is better designed for continuous control tasks like locomotion, and these results show that it does not translate well to Atari environments such as Space Invaders.

On the move_to_shield policy, we see that changing to A2C doesn't result in an improvement in policy likelihood from using TRPO (shown in Figure 3.2). The learning traces have a different quality; they fluctuate much more over time than the TRPO learning traces. This makes sense because A2C leads to more drastic policy changes at each time step than TRPO, which instead keeps changes within a "trust region" at each iteration. Changing ϵ doesn't have an obvious effect on the learning trace of the algorithm on this policy.

When learning the shield_shoot policy, however, we do see an improvement from using A2C as our generator algorithm. While the best policy log-likelihood that any of the TRPO-based trials achieves is less than -8, the best A2C-based trial achieves a much higher log-likelihood of -2.25. We also observe that the most successful trials were those with ϵ values of 0.025 and 0.065, and the trials with ϵ as 0 or 0.1 performed worse. This is evidence in favor of the hypothesis that a small amount of noise in the dataset helps GAIL better imitate the policy present in the data.

The fact that A2C-GAIL performs better on a more complex policy suggests that with human behavior it will continue to outperform TRPO by an even larger amount. The more complex policy of a human has a larger variation of states that it visits, and thus the discriminator is less strict as it allows more variation, which

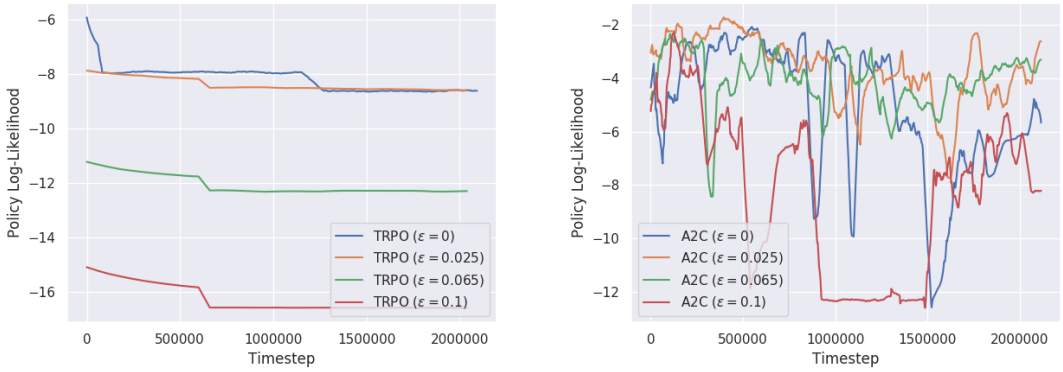

Figure 3.3: Learning shield_shoot with TRPO and A2C

in turn makes the generator's task easier. In contrast, these simple baseline policies are "clean" in a way that is uncharacteristic of humans, which makes them difficult to replicate in a way that isn't true of human data. This shows that the noisiness of real human data is a beneficial thing for this adversarial architecture.

Chapter 4

Quantitative Evaluation of Human Replication

Before moving on to modeling human behavior, it is necessary to first build a framework with which to evaluate the performance of various approaches. This chapter seeks to answer the question *how do we quantitatively measure the "humanness" of behavior?* It is notoriously difficult to evaluate and compare generative models [15]. To solve this problem, this chapter first identifies desirable properties of replicated behavior and different statistical measures of these properties. It then outlines a framework that enables easy comparison of generated behavior with a dataset, giving a rich set of statistical measures. This allows for interpretation of how successful various approaches are at imitating human behavior from a dataset, and it enables us to pinpoint which properties of the imitator's behavior need to improve.

4.1 Evaluation Framework

All of the measures described in this chapter are generalizable with respect to the data, so these methods are flexible in measuring imitation to diverse types of behavior. To best make use of these statistical measures, a computational framework is designed for easily comparing a set of trajectories to a target dataset of trajectories. Figure 4.1 describes this framework: it starts with either a learned policy or a dataset of behavior, and it represents the data as a sequence of actions since this is the minimal amount of information needed to encode the behavior. It then simulates the behavior in the environment to achieve a set of trajectories

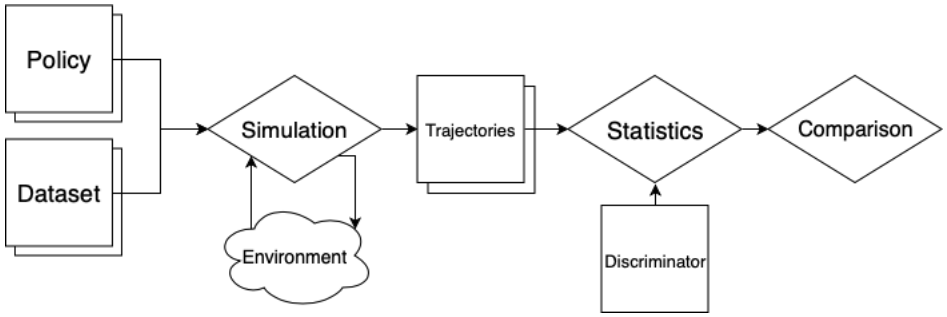

Figure 4.1: Evaluation framework

represented by tuples of state, action, and reward. It uses these trajectories to compute statistics such as the mean and variance of reward and episode length, as well as the mean discriminator reward. Using these quantities, it computes the statistical measurements outlined in this chapter to compare the behavior to the target data, which may or may not be human behavior.

4.2 Desirable Properties of Human Proxy

What are the properties of a human proxy that are necessary to consider it a replacement for a human? First, it must have a similar level of performance as the human; the issues of assuming super- or sub- human performance are discussed in Chapter 1. Not only that, but the manner in which it achieves this performance must be similar, as the state and action distributions displayed in the behaviors should be similar. While hard to define, it must also "look" human, not taking actions more frequently than a human would and not making un-human mistakes. Below are the statistical measurements that are used to quantitatively capture these traits.

4.2.1 Reward Distribution

One fundamental property that the learned agent must have is a similar reward distribution to that of the dataset. This captures the idea that a human proxy should play at a similar level of performance with respect to the environment as the human. To measure this, the mean and standard deviation of the total reward of the trajectories in the dataset are computed, and then the mean Z-score of the trajectories of our imitator is computed. The resulting quantity is the mean Z-score. Let R_i be the total reward for the ith trajectory for $1 \leq i \leq N$, and let R_H be the

distribution of human reward. The reward distribution score is defined as:

$$\text{Reward Distribution Score} = \frac{1}{N}\sum_{i=1}^{N}\frac{|R_i - E[R_H]|}{Var[R_H]} \tag{4.1}$$

4.2.2 Episode Length Distribution

Not only should the agent have a similar score at the end of the game, but we it also must to live for a similar amount of time. This captures how cautious or reckless the agent is when playing the game. The framework uses a similar approach to measure this, again computing the Z-scores with respect to the distribution of episode lengths in the dataset, using the mean of scores as the final result. Similar to Equation 4.1, let L_i be the episode length of the ith trajectory for $1 \leq i \leq N$, and let L_H be the distribution of human episode length. The length distribution score is defined as:

$$\text{Length Distribution Score} = \frac{1}{N}\sum_{i=1}^{N}\frac{|L_i - E[L_H]|}{Var[L_H]} \tag{4.2}$$

4.2.3 Action Distribution

It is also necessary to measure how similar the actions in the replicated trajectory are to the trajectories in the dataset. One way to do this is by comparing the distribution of actions, marginalizing over states. Let p and q be the action distributions of the human and the agent, respectively, such that $p(a) \in [0, 1]$ for all $a \in \mathcal{A}$ (and likewise for q). The Kullback - Leibler (KL) divergence, or relative entropy, of q from p is then measured:

$$D_{KL}(p||q) = -\sum_{a\in\mathcal{A}} p(a)\ln\left(\frac{q(a)}{p(a)}\right) \tag{4.3}$$

Thus, the lower this value, the more similar the action distributions are, which in turn means that the agent is more successful at replicating the human.

4.2.4 Action Pair Distribution

On another level, comparing two-step action distributions can capture more information such as how likely the agent is to change actions. This captures the

"smoothness" of the agent as well as other behavioral patterns for action pairs. To measure this, the framework computes the action pair distributions p_2 and q_2 for the human and the agent, respectively. Let $p_2(a_t, a_{t+1}) \in [0, 1]$ for all $a_t, a_{t+1} \in \mathcal{A}$, and likewise for $q_2(a_t, a_{t+1})$. q_2 and p_2 are empirically measured by observing the behaviors, and then the KL divergence of q_2 from p_2 is computed, or $D_{KL}(p_2||q_2)$ as defined in Equation 4.3. A low value for this quantity signals that the proxy is closer to replicating the true two-step action distribution of the human.

4.2.5 Discriminator Reward

Recall that GAIL yields both a generator that has been trained to imitate the behavior in the dataset and a discriminator that has learned to distinguish between human behavior and generated behavior. This discriminator can capture additional information that is relevant to measuring how similar the behavior of the imitator is to the human. An ideal discriminator uses all available information in the given state to determine if the state-action pair is from a human or not. In theory, the discriminator should learn to use this information to compute a measure that exactly captures the "human-ness" of a state-action pair, potentially including the previous statistical measurements as intermediate quantities. This suggests that the framework should look at the amount that the agent tricks the discriminator into thinking that it is human; the discriminator reward measures this quantity. As a result, the average discriminator reward is computed across all state-action pairs. Let M_i be the length of the ith trajectory for $1 \leq i \leq N$. Let s_{ij} and a_{ij} be the jth state and action of the ith trajectory, where $1 \leq j \leq M_i$.

$$\text{Discriminator Reward Score} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M_i}\sum_{j=1}^{M_i} -\ln(1 - D(s_{ij}, a_{ij})) \tag{4.4}$$

4.2.6 Predictive Accuracy

The probabilistic function of the learned policy can be used to measure how likely the human data is under this generative policy. This is measured as the mean likelihood of each state-action pair: $L = \frac{1}{M}\sum_{i=1}^{M} \ln(\pi(a_t|s_t))$ for dataset $\{(s_t, a_t)\}_{1 \leq t \leq M}$. Again defining s_{ij}, a_{ij}, and M_i as in Equation 4.4, the mean log policy likelihood is defined as:

$$\text{Log Policy Likelihood} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M_i}\sum_{j=1}^{M_i} \ln(\pi(a_{ij}|s_{ij})) \tag{4.5}$$

The higher (less negative) this value is, the more likely the data is given the policy, which indicates that the learned policy captures the policy distribution of human behavior.

These statistics as well as the described evaluation framework will be used to evaluate the human imitation algorithms in the next chapter.

Chapter 5

Replicating Human Behavior Via Hybrid Agents

This chapter describes the two main approaches to imitating human behavior building off of the work in previous chapters.

5.1 Failure Modes of GANs

Two of the biggest issues with training GANs are *generator collapse* where the discriminator is too accurate with respect to the generator and the generator is no longer able to learn, and *mode collapse* where the generator only produces samples at one of the modes of possible outputs and doesn't produce examples from the entire distribution [12; 8; 7]. The complex reinforcement learning environment of Space Invaders exacerbates these problems, leading to additional difficulties for the GAN-like algorithm of GAIL.

For one, if the generator produces one "bad" example action, this not only affects the current frame, but potentially the rest of the timesteps in the episode. For example, if the agent in the data never shoots but the generator shoots an alien, the discriminator will be able to see that an alien is gone and will know that all future state action pairs of the trajectory are fake, which prevents the generator from learning anything for the rest of the episode. In this sense, generator collapse is more extreme in an RL context as it affects not only single timesteps but potentially entire trajectories of actions.

Additionally, when modeling real human data, the generator agent can collapse

into a mode of the data and not explore much of the state space. For example in Space Invaders, this can happen when the agent only repeats actions that occur at the start of the trajectories in the dataset and stays towards the left of the screen where the human trajectories start. It only learns to replicate actions in that area of the screen, and while these are reasonable actions that a human would take, the lack of exploration of the state space is clearly sub-human behavior and would be unacceptable as a human proxy. Note that while the distribution of states would clearly differ from a human player, the GAIL discriminator is not able to make this distinction because it only looks at local state-action pairs and not the states of the entire trajectory. In some sense it is overfitting the frames at the start of the human trajectories and doesn't generalize well outside of that space.

To solve both of these issues, it is necessary to 1) allow the agent to still learn useful information even when the discriminator becomes too accurate and 2) incentivize the agent to explore more of the state space and achieve rewards from the environment. A natural way to do this is to find a method to incorporate true rewards from the environment into the algorithm.

5.2 Regularizing Against True Reward

To improve the performance of our human proxy, this chapter investigates building a system that takes into account both similarity to trajectories from the dataset and true rewards from the environment. By doing so, the system is able to have a notion of how to successfully play the game, but does so in a manner similar to that of a human. Just as regularizing the weights of a supervised learning model to some prior distribution helps prevent overfitting of the data, regularizing against the true rewards of the environment helps the learned agent act in a rational way that to some degree tries to maximize reward from the environment. Below are two methods for producing agents that have knowledge of the human data and the environment rewards.

5.3 Mixed Reward Function

One straightforward method to incorporate true rewards into A2C-GAIL is to perturb the reward function of this algorithm to capture information about true reward. Let $r(s, a) = -\ln(1 - D(s, a))$ be the reward function that the agent learns to maximize, where the reward is derived only from the degree to which the agent

fools the discriminator. Let $R(s, a)$ be the true reward experienced by the agent in the environment; the mixed reward function is defined as $\hat{r}(s, a) = r(s, a) + \beta R(s, a)$, where the constant β controls the coefficient of true reward. By doing this, the agent must learn to balance tricking the discriminator and playing the game in a way that achieves high reward. In practice, however, we see that the algorithm is very sensitive to the choice of β and that this perturbation complicates the learning process of the agent.

5.4 Hybrid Agents

For a given environment, there exist design continuums along which one can select agents to have certain properties. There is a spectrum of agent behavior between subhuman performance to superhuman, or (relevant in this case) a spectrum between a human replica that is too "overfitted" to the training data and an agent that only tries to maximize reward. This chapter uses a combination of the methods described earlier to construct a hybrid agent that is easily tuned to produce a replica of human behavior, avoiding the issues described earlier.

This approach makes the assumption that the true human policy can be expressed as a combination of two separate policies where one of the component policies is the optimal policy, and the other component is a "corrective" policy that captures human error. Given that human play is an approximation of optimal play, this is a natural way to break down the problem of building a human proxy into two independent subproblems. This insight means that using existing deep RL algorithms to train a well-performing agent is a useful component to build the human proxy.

5.4.1 Training Procedure

The goal is to build a hybrid that is capable of optimizing play in the environment and is also informed about human behavior from the data. To do so, an agent is trained to maximize performance in Space Invaders via A2C, where the agent only maximizes true reward and the discriminator has no effect on the reward function. After training this agent, a discriminator is then trained to discriminate between the performance of the actor-critic agent and the behavior in the dataset. This process, shown in Figure 5.1, produces an agent that can maximize performance and a discriminator that captures the difference in behavior between the agent and the human data.

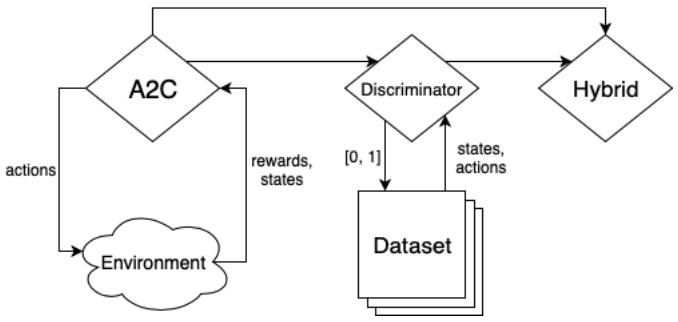

Figure 5.1: Hybrid Agent Training

5.4.2 Hybridization

This section describes how to combine the information from the A2C agent and the discriminator to produce a hybrid that actively uses information from both. This policy must be tunable so that it can find the right balance between the two extremes.

For a given state s, the hybrid policy is structured as follows. It first computes the distribution over actions from A2C using the learned policy $\pi(a|s)$. It also computes the reward from the discriminator for each possible action from the given state using the learned discriminator function: $r(s, a) = -\ln(1 - D(s, a))$. Note that the discriminator rewards for all actions from a given state do not necessarily sum to one. It then multiplies the values from the A2C policy by a tunable parameter λ and adds the discriminator rewards, normalizing over actions so that it is a probability distribution. The resulting distribution, $\pi_\lambda(\cdot|s)$, is the hybrid policy function:

$$\pi_\lambda(a_i|s) = \frac{\lambda\pi(a_i|s) + r(s, a_i)}{\sum_j [\lambda\pi(a_j|s) + r(s, a_j)]} \tag{5.1}$$

This method, shown in Figure 5.2, is expressive in the sense it is possible to construct a discriminator function $D(s, a)$ such that $\pi_\lambda(s, a)$ is arbitrarily close to any distribution. This is important because it means that by incorporating the policy from A2C, it does not limiting the set of policies the agent can learn. Also note that this method does not normalize the reward from the discriminator before adding the A2C policy; this is because the magnitude of these rewards matter. For example, when the discriminator rewards are low no matter which action the agent picks, the agent should weight the A2C policy relatively more heavily since any action in that state won't appear very "human," so it should play in a rational way. Likewise, when the discriminator reward for a certain action is high, it should

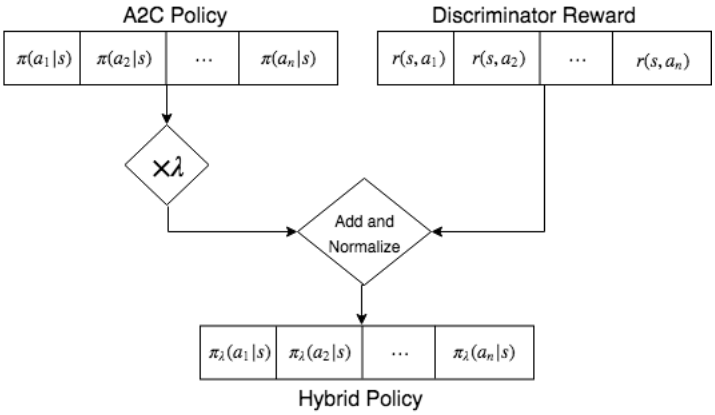

Figure 5.2: Hybrid Policy

weight that action higher than the A2C policy since we want to choose actions that are highly "human". The parameter λ controls for the relative strength of the A2C policy.

5.5 Results

The comparison and evaluation framework from Chapter 4 is now applied to the human imitation approaches defined above.

5.5.1 Human Data and GAIL Baseline

A sample of 39 episodes of human play in Space Invaders was collected from 12 subjects. The summary statistics of these data are shown in Table 5.1. A2C and A2C-GAIL were also trained on this dataset as baselines, and the performance statistics are shown in this table as well. As these numbers demonstrate, GAIL wasn't able to appropriately achieve the performance level of the human data, and A2C performs at a significantly superhuman level. This validates our intuition that

	Mean Episode Score	Mean Episode Length
Human	4.67	902.03
A2C-GAIL	-11.90	426.30
A2C	33.20	1093.80

Table 5.1:: Human Data, A2C-GAIL, & A2C Statistics

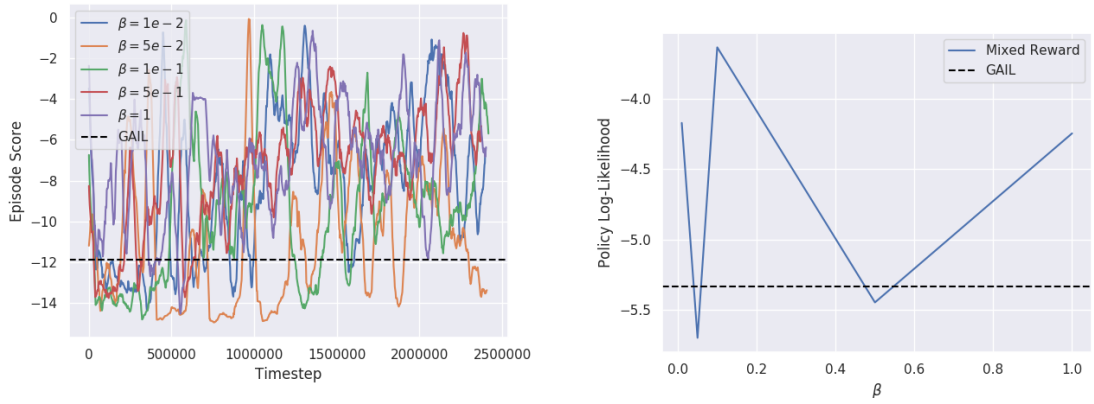

Figure 5.3: Performance of Mixed Reward Functions

we need to utilize reward from the environment in some way so that our human imitator can achieve a similar level of play to the data.

5.5.2 Mixed Reward Function

The approach where we add a coefficient of true reward to the reward function from GAIL is evaluated first. Using the collected human behavior data, agents were trained using A2C-GAIL for around 2.5×10^6 timesteps each, using β values of 0.01, 0.05, 0.01, 0.1, 0.5, and 1. Figure 5.3 shows the results of these experiments.

On the left, we see that higher values of β lead to higher episode score, and by mixing in the true environment reward we get higher scores than with GAIL alone. However, none of the trails achieved a score close to the human level. This was even true at $\beta = 1$, when the reward from the discriminator and true reward from the environment are valued equally. This suggests that the mixed reward function was difficult to maximize and the current approach struggles to perform well in this environment.

The right of Figure 5.3 shows the learned policy log-likelihood plotted as a function of β. We see that the log-likelihoods tend to be higher than that of GAIL; this shows that incorporating knowledge about rewards from the environment can lead to a better fit of the human data. However, there doesn't appear to be an obvious trend of the likelihood with respect to β, and the likelihoods are still too small to be a useful human imitator. This shows that we need an approach that can perform better in the environment and can also better match the data of human play.

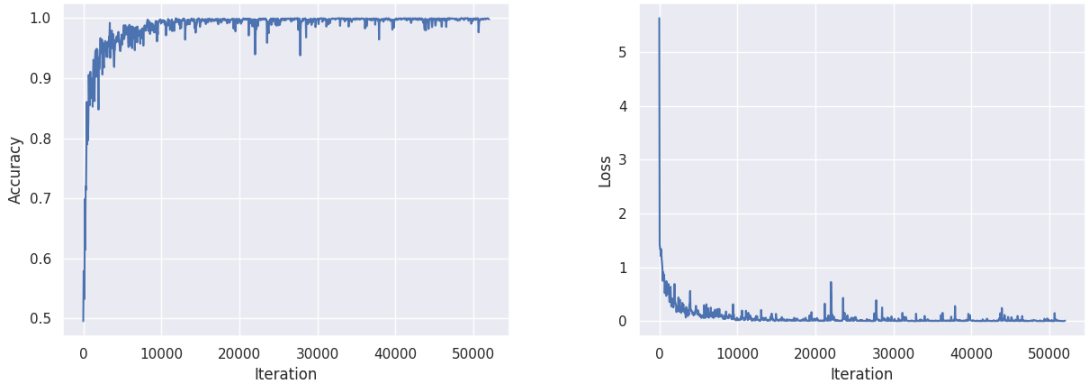

Figure 5.4: Training Discriminator

5.5.3 Training Discriminator

The Hybrid Agent approach uses an agent that performs well in the environment and a discriminator that can accurately predict whether a state-action pair is from a human or from the agent. A2C is used to train an agent that has a level of performance well above the humans in our data; the summary statistics of this agent's performance are listed in Table 5.1.

Supervised machine learning is then used to train a discriminator to predict the probability that a state-action pair is from the human data or from the AI agent. The loss function used for this algorithm is the mean squared error of predictions. Figure 5.4 shows the training process for the discriminator. On the left, the mean accuracy of the discriminator at each iteration is plotted, showing that the discriminator has more or less converged after 30,000 iterations. On the right, the prediction loss (mean squared error of predictions) at each iteration is plotted; again we see that the loss stabilizes to a low value after 30,000 iterations. These results show that there is an inherent difference between the way the AI agent plays the game and how humans play the game, as our discriminator is able to learn to identify behavior as human or artificial with an accuracy of nearly 100%.

5.5.4 Tuning & Evaluating Hybrid

Hybrids are generated with 51 values of the parameter λ (from Figure 5.2) linearly spaced from 0.0 to 2.0. These hybrid agents are then evaluated using the framework from Chapter 4. Below the results shown in Figures 5.5 to 5.10 are discussed:

- In Figure 5.5, we see that the policy log-likelihood decreases nearly linearly with respect to λ. This makes sense because higher values of λ will weight the policy from A2C more heavily, causing it to perform at a higher level but in a less human way. For all of the values of λ, the policy log-likelihood is significantly higher than that of either GAIL or A2C. This plot shows that the hybrid approach is a significantly better fit for the human data and that our parameter λ can be used to make a tradeoff of how closely the hybrid policy fits the data.

- Figure 5.6 shows that the hybrids achieve a much higher mean discriminator reward than GAIL and A2C. This is to be expected since the hybrid policy takes this discriminator into account. There isn't a clear trend in the data, but the higher amount of reward from the discriminator shows that the hybrids are able to "fool" the discriminator much more effectively than GAIL or A2C.

- In Figure 5.7, we see that the mean episode score of the Hybrid Agents is centered around the mean score of the human data, with some variance. This is a success; the hybrid imitator is able to play at a similar level as the human, which is a necessary property of a human proxy. For small values of λ, the agent performs at a sub-human level which shows that it is necessary to incorporate environment reward by having a non-zero λ. As expected, the mean score of A2C is greater than both the human data and the hybrids. GAIL achieves a similar mean episode reward to the human, but slightly lower.

- Figure 5.8 shows that the mean reward Z-score for the Hybrid Agents is between those of GAIL and A2C, with a slight upward trend. Combined with the results in Figure 5.7 this shows that GAIL deviates the least from the level of human performance, the hybrids deviate slightly more, and A2C deviates the most.

- In Figure 5.9, we see that the action pair KL-divergence increases with λ and is lower than that of both GAIL and A2C for $\lambda < 1.5$. This shows that the action distributions of the Hybrid Agents resembles that of the human data much more accurately than A2C or GAIL for this range of λ. It is intuitive that increasing λ leads to a higher KL-divergence since it weights the policy from A2C more heavily, which leads to less human-like behavior.

- Figure 5.10 shows that the episode length distribution deviates more from the human data than does A2C or GAIL, however not by a large amount. The Hybrid Agents tend to have shorter episodes than those of the human data. It is interesting that the hybrids achieve a similar score and action distribution,

but do so in a shorter amount of time. A potential future area of work could be to make this algorithm more closely fit the episode length distribution of the data.

Appendix A shows a table of comparison statistics that were computed by the Chapter 4 framework and used to generate the above figures.

These results show that the Hybrid Agent approach demonstrates a significant improvement over previous approaches across nearly all of the comparison metrics. This is promising since the produced agent behaves in a statistically similar way to the human, which will most likely be useful when training a Helper-AI to cooperate with it.

Another benefit of the Hybrid Agent approach is that it is fast to tune and iterate on, which is a useful property for the Helper-AI training process shown in Figure 1.1 and discussed in Chapter 1. First, training the discriminator is a much less computationally difficult task than training an agent to maximize score in Space Invaders. As a result, we see that the discriminator converges after a relatively short period of time (30,000 iterations in Figure 5.4). This means that each time we need to train a Hybrid Agent on a new dataset of human behavior, this process won't take very long since the discriminator converges quickly. Second, once a well-performing A2C agent is trained, it can be reused for all subsequent Hybrid Agents; all that is required is to train a new discriminator using the same A2C agent. Third, once we have a trained discriminator, it can be used to produce a hybrid with an arbitrary value of λ "for free". This means our hybrid's behavior can be easily tuned to find the best balance between maximizing performance and imitating the human data.

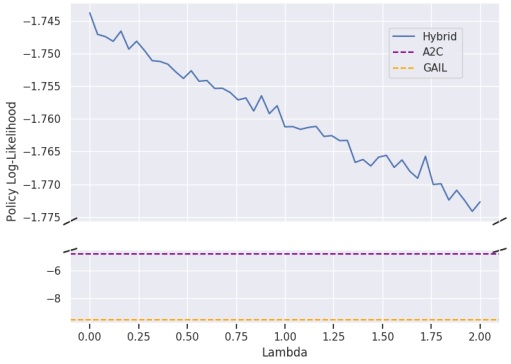

Figure 5.5: Policy Likelihood

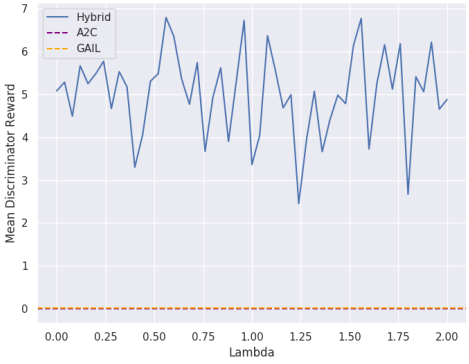

Figure 5.6: Discriminator Reward

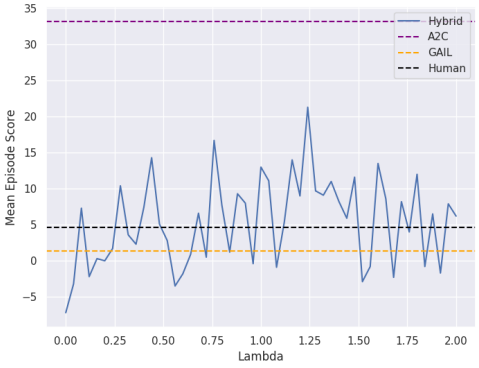

Figure 5.7: Episode Score

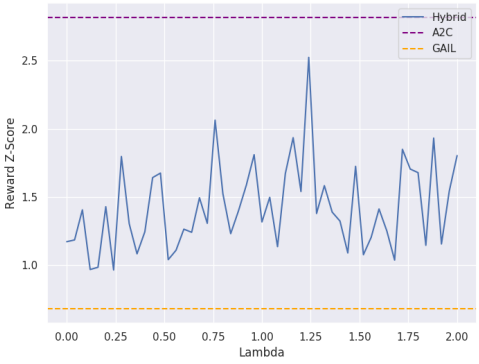

Figure 5.8: Reward Z-Score

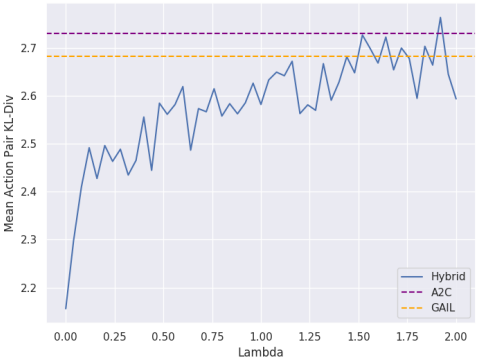

Figure 5.9: Action Pair KL-Div

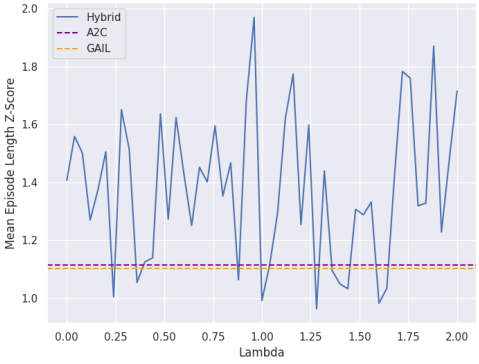

Figure 5.10: Length Z-Score

Chapter 6

Conclusion

Throughout this thesis, the strengths and limitations of various approaches to human imitation have been investigated as well as the unique challenges presented by high-dimensional environments such as Space Invaders. GAIL, a state-of-the-art deep imitation learning algorithm, was modified to work in Atari environments and the modified algorithm was tested it to show that it scales up to more complex environments more effectively than the original version of the algorithm. However, the results show that it is difficult to scale up existing imitation learning algorithms to work in this more complex environment. Specifically, issues with the generative adversarial architecture of GAIL, such as generator collapse and mode collapse, make it challenging to imitate the human policy while still achieving a similar score as the human data.

This thesis also described a framework for evaluating and comparing human imitators based off of a set of relevant statistics, providing a computational procedure for computing these comparisons in an efficient and flexible way. This framework was used to test the claims and justify the design choices made in this thesis. By using this framework, the original and modified versions of GAIL were shown to have insufficient performance and imitation of human behavior to provide an effective human proxy.

However, Chapter 5 demonstrated that it is possible to overcome these challenges by regularizing against true reward from the environment, doing so in a modular, hybrid way. The best-performing contribution of this thesis is a novel human imitation approach that combines reward-maximizing behavior and knowledge of human behavior from the dataset. This approach is a better imitator for the human data across nearly all of the metrics than GAIL and the previous approaches tested in the thesis.

6.1 Future Directions

These contributions provide interesting directions for future research. A natural first application of this work is to use the Hybrid Agent approach in the Helper-AI training procedure discussed in Chapter 1. This would allow us to measure how much using this human proxy would improve joint score in cooperative tasks. The effects of the tuning parameter λ on performance and other properties of cooperative play can also be investigated.

To build a Helper-AI that is robust to various skill levels of partners, the human data can be segmented by skill level and a set of different human imitators can be trained using the approaches of this thesis. These imitators will behave in different ways, and by training a Helper-AI against all of them, the trained agent will be able to cooperate well with a variety of different partners.

Another interesting component of the Hybrid Agent approach is that the discriminator has learned, in some sense, the pattern of human error. While this thesis only uses the discriminator to build a human imitator, it can also be used to measure and better understand how humans deviate from optimal performance. One hypothesis is that humans play worse in more "stressful" situations, possibly where state values are low and where there is a high likelihood of losing a life. This hypothesis can now be tested using the human imitator by measuring the KL-divergence of the human proxy policy and the policy of a well-performing agent as a function of state value. We can then investigate these results to see how strongly state value affects performance.

An interesting implication of this work is that it can be used to make live predictions of human performance as a human engages in a task. This allows for products that can warn a human when they are in a state space where they are likely to make a mistake. It is also possible to build an "intervention AI" that can predict when the human is in a state space where they are likely to behave suboptimally, and it can then intervene to take over for a short while at some cost [11].

In summary, this thesis has shown that it is possible to model and imitate human behavior in an environment as complex and high-dimensional as Space Invaders, and we can use the approaches outlined in this thesis to build AI systems that effectively cooperate with humans, understanding their flaws and maximizing joint performance in a way that would not be possible without a model of human behavior.

Appendices

Appendix A

Complete Results

This appendix contains a table of the comparison results from the Hybrid Agent in Chapter 5.

Name	Policy Likelihood	Reward Z-Score	Episode Score	Length Z-Score	Discriminator Reward	Action Pair KL-Div
a2c_model	-4.77972	2.8165	33.2	1.11483	0.00231342	2.73048
hybrid_lam_0.000_s	-1.74375	1.17135	-7.2	1.40722	5.08242	2.15605
hybrid_lam_0.080_s	-1.74738	1.40496	7.3	1.50127	4.49132	2.40947
hybrid_lam_0.160_s	-1.74654	0.983802	0.3	1.37512	5.25002	2.42748
hybrid_lam_0.240_s	-1.74808	0.96406	1.7	1.00492	5.77154	2.46312
hybrid_lam_0.320_s	-1.75107	1.30296	3.6	1.5151	5.52948	2.43462
hybrid_lam_0.400_s	-1.75161	1.24374	7.4	1.1255	3.30222	2.55552
hybrid_lam_0.480_s	-1.75379	1.67477	5.1	1.63675	5.30675	2.58446
hybrid_lam_0.560_s	-1.75422	1.10883	-3.5	1.62437	6.79383	2.5816
hybrid_lam_0.640_s	-1.75532	1.24045	0.9	1.25135	5.35763	2.48657
hybrid_lam_0.720_s	-1.75595	1.30625	0.5	1.40203	5.74207	2.56655
hybrid_lam_0.800_s	-1.75678	1.52341	7.7	1.35337	4.93273	2.55758
hybrid_lam_0.880_s	-1.75645	1.39838	9.3	1.06414	3.90177	2.56226
hybrid_lam_0.960_s	-1.75798	1.80967	-0.4	1.96943	6.72612	2.62607
hybrid_lam_1.040_s	-1.76117	1.49709	11.1	1.11653	4.04588	2.63288
hybrid_lam_1.120_s	-1.76129	1.67148	5.4	1.62123	5.56308	2.64164
hybrid_lam_1.200_s	-1.76267	1.53986	9	1.25444	4.99194	2.56275
hybrid_lam_1.280_s	-1.76331	1.37864	9.7	0.96417	3.95315	2.56961
hybrid_lam_1.360_s	-1.76665	1.38851	11	1.09552	3.66257	2.59049
hybrid_lam_1.440_s	-1.7672	1.08909	5.9	1.03313	4.98112	2.6813
hybrid_lam_1.520_s	-1.76558	1.07593	-2.9	1.28827	6.13521	2.72655
hybrid_lam_1.600_s	-1.76628	1.41154	13.5	0.983147	3.72604	2.66811
hybrid_lam_1.680_s	-1.76908	1.03645	-2.3	1.42926	6.15837	2.65395
hybrid_lam_1.760_s	-1.77003	1.70438	4	1.75981	6.18236	2.67785
hybrid_lam_1.840_s	-1.77242	1.14503	-0.8	1.3286	5.41162	2.70298
hybrid_lam_1.920_s	-1.77244	1.1549	-1.7	1.22862	6.2183	2.76307
hybrid_lam_2.000_s	-1.77269	1.80309	6.2	1.7157	4.88072	2.59342

Table A.1:: Hybrid Comparison Results

Appendix B

Technical Details

This appendix describes some of the implementation details of the various algorithms used in this thesis. The implementations of many of these algorithms build off of the OpenAI Baselines framework.

B.1 Space Invaders Environment Details

The Space Invaders environment referred to throughout this thesis uses the same wrapper as in the DQN paper [10], with a few modifications:

- There are only two possible reward values; +1 for whenever the agent shoots an alien and -5 when the agent loses a life. Note that originally, the agent would only experience a negative reward when it lost 3 lives, and this reward was -1. This change was made so that the agent properly learned to avoid bullets.
- The environment only changes action every 4 frames, repeating the chosen action for the 3 frames in between. This is helpful to speed up the training process.
- Each pixel values from the screen is converted to a single greyscale value. The screen is downscaled to 84 by 84 pixels.
- The state representation shows the current screen's pixel values as well as the past 3 screens. The complete state array's dimensions are 84x84x4.
- The environment's seed, used for the random number generator, was always set to 0 throughout the experiments in this thesis.

B.2 TRPO-GAIL Details

For each training batch of the TRPO generator, the policy was run for 20 steps in the environment. The maximum KL divergence was $\delta = 0.01$, and the discount rate $\gamma = 0.995$. It used an Adam Optimizer to minimize the loss function of both TRPO and the discriminator [6].

Each iteration of TRPO-GAIL consisted of 3 iterations of training the generator and 1 iteration of training the discriminator. For each training iteration of the discriminator, it randomly sampled the same number of state-action pairs from the dataset as it received from the generator, in this case this was 20 * 3 = 60.

B.3 A2C Details

The A2C implementation used in this thesis was the same for both standalone A2C and A2C-GAIL, with the difference of its reward function. We used 4 environments running simultaneously so that at each step we get 4 states and 4 actions. For each training iteration, we run the policy for 20 time steps in the environment to sample states, actions, and rewards. The complete loss function used was the sum of the policy loss and 0.5 times the value function loss (these losses are defined in Chapter 2). The coefficient on entropy for the policy was $\beta = 0.01$, and the discount rate was $\gamma = 0.99$. It used an Adam Optimizer to minimize the loss function at each iteration. This algorithm was trained on GPU-accelerated hardware. In A2C-GAIL the details of training the discriminator were the same as for TRPO-GAIL.

B.4 Training Discriminator Details

When training the standalone discriminator for the Hybrid Agent, the training process used a batch size of 240 state-action pairs from the agent and 240 state-action pairs from the dataset at each iteration. The pairs from the dataset were sampled uniformly at random. The loss function was the mean squared error of predictions, and this loss function was minimized using an Adam Optimizer.

References

[1] Open AI. 2018. OpenAI Gym. *https://gym.openai.com* (15 Dec. 2018).

[2] Thomas Degris, Martha White, and Richard S. Sutton. 2012. Off-Policy Actor-Critic. *CoRR* abs/1205.4839 (2012). http://arxiv.org/abs/1205.4839

[3] Christos Dimitrakakis, David C. Parkes, Goran Radanovic, and Paul Tylkin. 2017. Multi-View Decision Processes: The Helper-AI Problem. In *Proc. 30th Advances in Neural Information Processing Systems (NIPS'17)*. 5449–5458. https://econcs.seas.harvard.edu/files/econcs/files/dimitrakakis_nips17.pdf

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[5] Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. *CoRR* abs/1606.03476 (2016). http://arxiv.org/abs/1606.03476

[6] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

[7] Lars M. Mescheder. 2018. On the convergence properties of GAN training. *CoRR* abs/1801.04406 (2018). http://arxiv.org/abs/1801.04406

[8] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled Generative Adversarial Networks. *CoRR* abs/1611.02163 (2016). http://arxiv.org/abs/1611.02163

[9] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. *CoRR* abs/1602.01783 (2016). http://arxiv.org/abs/1602.01783

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (25 02 2015), 529 EP –. https://doi.org/10.1038/nature14236

[11] David C. Parkes, Paul Tylkin, and Goran Radanovic. 2019. Multiplayer Atari: A Framework for Scaling Up Helper-AI. *"Under Review"* (2019).

[12] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. 2016. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 2234–2242. http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf

[13] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust Region Policy Optimization. *CoRR* abs/1502.05477 (2015). http://arxiv.org/abs/1502.05477

[14] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction*. MIT Press. http://www.worldcat.org/oclc/37293240

[15] Lucas Theis, Aäron van den Oord, and Matthias Bethge. 2015. A note on the evaluation of generative models. *arXiv e-prints* (Nov 2015), arXiv:1511.01844.

[16] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral Cloning from Observation. *CoRR* abs/1805.01954 (2018). http://arxiv.org/abs/1805.01954

[17] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *CoRR* abs/1708.05144 (2017). http://arxiv.org/abs/1708.05144