Learning to Imitate, Adapt and Communicate

Siddharth Agrawal

CMU-RI-TR-21-38

August 12th, 2021

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Masters in Robotics.

To my family.

Abstract

For AI agents to co-exist with humans, they need to be able to learn from us humans, adapt to any perceived changes in our behavior , and communicate in a manner that is easily interpretable. In this work we investigate the following 3 subproblems: Imitation Learning, Adaptation in Human-Agent Teaming and Learning Interpretable Communication. First, we study Generative Adversarial Imitation Learning, a state of the art imitation learning approach. We discuss its limitations theoretically as well as empirically. Particularly, we look at the reward bias issue in GAIL and propose neutral rewards for overcoming the reward bias problem. Next, we demonstrate how our method out-performs the existing approaches. Then, we study the problem of adaptation in context of Team-Space-Fortress, a 2 player game. We design an agent library, which tries to capture the space of human policies, and create a set of similarity metrics which could evaluate the similarity of human policies to policies in the agent library. We leverage this similarity metric to adapt the policy of the AI agent, partnered with a human, to improve overall team performance. For efficient human-agent teaming it is also crucial that agents communicate in a manner that is interpretable to humans. Humans compose a finite vocabulary of words to communicate. Further, they also communicate at a rate which another human could understand. Therefore, for AI agents to be good partners to humans, they need to be able to learn communication protocols that use discrete messages instead of continuous vectors and they need to learn when to communicate and at a rate which humans would be able to understand i.e the communication needs to be sparse. In this work, we first identify the shortcomings of existing approaches for learning communication sparse protocols using multi-agent reinforcement learning. Then we propose a method capable of learning communication protocols that are sparse and use discrete tokens.

Acknowledgments

I would like to thank my advisor Prof. Katia Sycara for providing me the opportunity to explore problems in the areas of my interest. Her technical guidance has been extremely crucial for the progress I was able to make in each of the research problem that I worked on over the course of 2 years. Her work ethic and enthusiasm for conducting research served as a constant inspiration to me. I would like to thank Dana Hughues, Prof. Michael Lewis and Prof. Julie Shah for providing crucial advice for my research at various times during the 2 years.

I would also like to thank my collaborators, Rohit Jena, Tianwei Ni, Huao Li, Mycal Tucker, Suhas Raja, Yikang Gui. I have learnt a lot from their technical expertise over the course of 2 years. I also want to acknowledge the contributions of Swaminathan Gurumurthy, Tejus Gupta, Vidhi Jain, Ankur Deka with whom I have had really interesting discussions about the areas of my interest and they have also been a great source of inspiration for my research work. Finally, I am grateful to my family and friends for always being there for me and believing in me.

Funding

This research was supported by ARL awards W911NF19-2-0146 and W911-20-2-0081 and AFRL/AFOSR award FA9550-18-1-0251

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Addressing Reward Bias in Generative Adversarial Imitation Learning

1.1 Introduction

Adversarial Imitation Learning(AIL) algorithms [27] [20] [31] [35] have been shown to achieve state of the art performance on a variety of imitation learning tasks. Performance of AIL algorithms have been primarily evaluated in two kinds of broad settings. The first kind of setting can be characterised as survival based environments. For example, environments in Mujoco like Hopper, Cheetah where the agent is rewarded to survive in the environment by remaining in a set of good states and penalized (by negative rewards or by termination) if it attains states other than these good states. For example, in Hopper the agent gets good rewards for hopping forward as fast as possible. If the agent gets into a set of 'bad poses' the episode is terminated. The other setting can be characterised as "task" based environments where the agent is rewarded to complete a task (reaching a goal state, interacting with an object in the environment, dodging an obstacle) and so on. The episode terminates once the agent completes the task. However, there is another important setting which hasn't drawn much attention in the Imitation Learning community. The episode may also terminate when the agent has failed to perform the task in a task based scenario. For example, when the agent may alter the environment such that completing the task becomes impossible, such as breaking a glass that the agent was supposed to pick up. These are environments where the agent is rewarded to complete a task, i.e to reach a certain state in the shortest amount of time after which the episode is terminated, however, there are some additional bad terminal states, upon reaching which the episode terminates and the agent doesn't receive a reward. Most of the community has worked on environments which are either purely survival based or purely task based. However, to show that current reward functions are biased, we need to use more complex environments. To that end, we use different environments of the Gym-Minigrid[14] package and LunarLander. The environments have tasks with single and multiple terminal states, which would be ideal to test if learning is hindered by the inherent biases in the reward function itself.

Our experiments show that existing methods[27][31] often get stuck in a suboptimal policy in task based environments with multiple terminal states. We provide a rough theoretical sketch of the underlying issues in these methods. We also propose a reward function which is able to achieve comparable results in task based environments with single terminal state and performs significantly better than existing methods in task based environments with multiple terminal states.

1.2 Related Work

Early works in Imitation Learning based on Inverse Reinforcement Learning [1] focus on recovering a reward function from expert demonstrations which is optimised using Reinforcement Learning. These methods are slow since they require solving the RL objective in the inner loop. From a practical standpoint, the intermediate step of recovering reward function is not really necessary when the goal is to only learn to imitate the expert.

Recent works in Adversarial Imitation Learning [27] [20] [41] [29] [17] focus on directly recovering the expert policy without the need for recovering the exact reward function. These methods alleviate the problem of distributional shift faced by some of the more traditional approaches like Behavior Cloning and its variants [63] [50] [49] [15]. GAIL based methods are also very sample efficient than behavior cloning in terms of number of expert trajectories required to recover the expert policy. Although AIL methods have had a fair amount of success in various environments, recent work [31] has highlighted the implicit bias in the formulation of its reward function which results in degradation of performance in certain environments. The work in Discriminator-Actor-Critic [31] show how the reward function used in GAIL for training the agent is biased which leads to degradation of performance in certain environments. In GAIL, the agent focuses on how to maximize its rewards from the discriminator, which may not necessarily translate to following the expert policy. In task-based environments, the problem of implicit bias due to zero rewards for terminal states encourages the agent to stay in the environment to collect more positive rewards, hence ignoring the task. Their method worked on the survival based agents, as well as on task-based robotic environments. However, there are no results on the method's performance in task based environments with multiple terminal states. Our experiments show their formulation doesn't work well in such task based environments with multiple terminal states. To the best of our knowledge, [31] is the only work which explicitly addresses reward bias and proposes a way to solve it.

1.3 Theory

1.3.1 Generative Adversarial Imitation Learning

We make use of the GAIL[27] framework to learn the expert policy. In the GAIL framework, a discriminator(D) is trained to differentiate between transitions from the expert and from the trained policy. The policy is trained using an on-policy RL algorithm, with the reward function being the probability of the action being equal to expert action in a given state. The GAIL objective can be written as:

$$\max_{\pi} \min_{D} \left[E_{a\sim\pi(s)}[\log(D(s,a))] + E_{a\sim\pi_E(s)}[\log(1-D(s,a))] + \lambda H(\pi) \right] \tag{1}$$

where π is the imitation learning policy, $H(\pi)$ is the entropy of the policy, π_E is the expert policy and $D(s,a)$ is the probability that the state-action pair is from the expert trajectory output by the discriminator. GAIL does not aim to recover the true reward function, but rather tries to match the state occupancy measure of the expert and the agent. However, this does not guarantee that the expert trajectories are optimal with respect of the reward function output due to biases in the choice of reward function.

1.3.2 Survival Based and Task Based Environments

In survival based tasks, the agent gets a reward for staying in a set of ”good states” and the episode terminates when the agent goes to a ”bad state”. Within the good states, different rewards may be provided to encourage the agent to stay within certain good states more than others. Some examples of these kinds of environments are the Mujoco environments like Hopper, Reacher etc.

In task-based environments, the agent gets a reward for successfully completing a task, characterised by agent reaching a certain state in the environment. Additional rewards may be given for completing ‘subtasks’ along the way. The agent may also be penalized for simply existing in the environment. This makes the agent complete the task as soon as possible to get maximum positive rewards from task completion and minimum survival penalty.

Additional complexities might arise, if additionally the environment also has some other terminal states, which don’t lead to completion of the task but lead to the termination of the episode (the agent essentially dying). In our work, we use 4 environments of this kind and show how existing approaches[31] [27] don’t perform optimally on these environments and how our method tackles the biases.

1.3.3 Reward bias in GAIL

To train the imitation policy in the GAIL method, two kinds of discriminator reward are commonly used in literature [17], [27] :

1. **Positive Reward**: This reward is always positive and its value is $-\log(1 - D(s, a))$
2. **Negative Reward**: This reward is always negative and its value is $\log(D(s, a))$

However, both the reward functions have implicit biases which may hinder learning. They are explained in the following subsections.

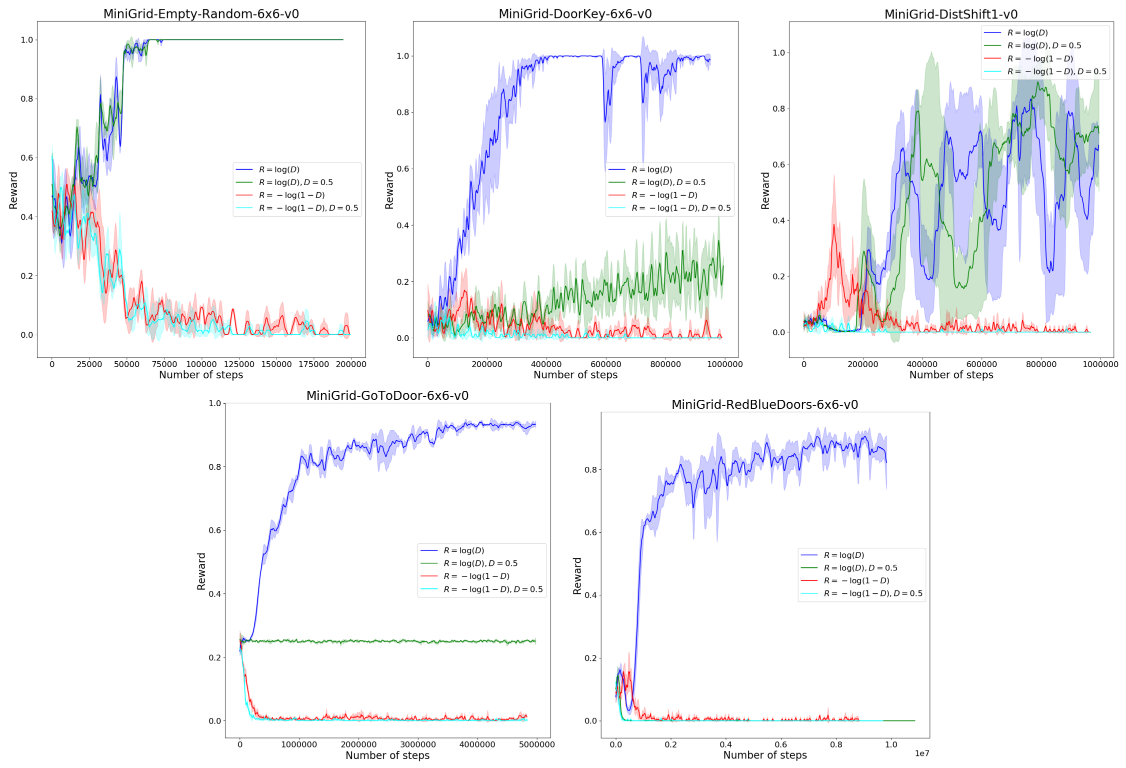

Figure 1.1: Episode rewards for different environments and different GAIL reward functions. The positive rewards show a survival bias because they do not complete the task (leading to low reward). The negative rewards show a termination bias, and in the simpler environments, the agent learns to complete the task even without trajectories ($D = 0.5$) only due to the reward bias.

Survival bias in positive rewards

In a task-based environment, providing non-negative rewards at every step may lead to survival bias and the agent may not complete the task and loop in the environment to collect more rewards. To show that survival bias is indeed an issue, we consider an ideal case. We consider an "oracle discriminator", which gives a positive reward R when the agent performs the expert action for a given state, and 0 otherwise. In other words, the reward function of the discriminator is $R(s,a) = R$ if $a = \pi_E(s)$ and $R(s,a) = 0$ if $a \neq \pi_E(s)$. In practice, the discriminator reward is also bounded to prevent numerical issues by using $R(s,a) = -\log(\max(\epsilon, 1 - D(s,a)))$ which has range $\left[0, \log(\frac{1}{\epsilon})\right]$. Now consider an expert trajectory of path length p. Since the discriminator will give a positive reward R at each step in the expert trajectory, the total discounted reward of the expert trajectory is:

$$R_E = R + \gamma R + \ldots \gamma^{(p-1)} R = \frac{1-\gamma^p}{1-\gamma} R \tag{1.1}$$

Now consider a trajectory where the agent follows the expert trajectory for $p-1$ steps, and then loops between an expert action and non-expert action to survive for as long as possible. The total discounted reward for this trajectory is:

$$\begin{aligned} R_{loop} &= R + \gamma R + \ldots \gamma^{(p-2)} R + 0 + \gamma^{(p)} R + 0 + \gamma^{(p+2)} R \ldots && (1.2) \\ &= \frac{1-\gamma^{(p-1)}}{1-\gamma} R + \frac{\gamma^p}{1-\gamma^2} R && (1.3) \end{aligned}$$

For the agent to prefer this trajectory over the expert trajectory, R_{loop} should be greater than R_E. This inequality will imply the following:

$$\begin{aligned} \frac{1-\gamma^{(p-1)}}{1-\gamma} R + \frac{\gamma^p}{1-\gamma^2} R \geq \frac{1-\gamma^p}{1-\gamma} R && (1.4) \\ \implies \frac{\gamma^p}{1+\gamma} \geq \gamma^{(p-1)}(1-\gamma) \implies \gamma \geq 1 - \gamma^2 && (1.5) \end{aligned}$$

which is true for $\gamma \geq 0.6180$. In practical scenarios, $\gamma \approx 1$, and hence, the agent can prefer to loop in the environment rather than following the expert trajectories. It is therefore evident as to why positive rewards might be unsuitable for task based environments.

Termination bias in Negative Rewards

Positive reward functions are prone to survival bias, and the next obvious choice of a reward function is $R(s,a) = \log(D(s,a))$. This reward function is always negative since $D(s,a) \in [0,1]$. However, this reward function makes an agent prone to termination bias. In termination bias, the agent tends to end the episode as soon as possible to stop accruing more negative rewards. This may be beneficial for task based environments where the agent wants to complete the task as soon as possible. However, if there are faster ways to terminate an episode, the agent may be biased to do that instead of completing the task. In our experiments, we show that GAIL with the reward function $R(s,a) = \log(D(s,a))$ fails to learn the task when there are multiple termination conditions in the environment.

1.3.4 Discriminator Actor Critic

Both the survival and task based environments suffer from different problems but for the same reason - the reward for the terminal state is implicitly set to 0 in all these problems, which is why positive rewards become unsuitable for task based environments and negative rewards become unsuitable for survival based environments.

To address the issue of implicit bias in the reward function of the GAIL, [31] propose learning rewards for absorbing state explicitly. Consider a trajectory of T time steps, let the reward for the terminal state i.e the Tth step be $R_T = R(s_T, a_T)$, in the standard GAIL setting as in Equation 1. In DAC, a new reward function for terminal state is defined $R_T = R(s_T, a_T) + \Sigma_{T+1}^{\infty} \gamma^{t-T} R(s_a, .)$ where $R(s_a, .)$ is learnt. This learnt reward for the terminal state removes the bias towards avoiding or transitioning to the terminal state. With this formulation, GAIL tries to match the state occupancy of the expert and agent for the terminal state as well, hence mitigating survivor bias. However, this modifies a finite-horizon environment into an infinite-horizon environment, and does not consider the scenario where the agent may terminate an episode in a multitude of different ways. There are a few other problems as well that the work does not address:

1. **Requires modifying the environment**: In the original implementation

of DAC, wrappers are provided to augment the environment with an extra terminal state. However, this may not always be feasible as we may only have an immutable API to the environment. This may also be a problem in real environments.

2. **Does not handle multiple terminal states**: In a task-based environment with multiple terminal states, the agent can terminate an episode in multiple ways, out of which only some may correspond to task completion, and this can hinder learning. If a large portion of the expected reward comes from being in the terminal state, then the agent may not distinguish between expert trajectories and other similar trajectories which terminate the episode, as long as the state occupancy of the terminal states match for the agent and the expert.

3. **May be sample inefficient**: In the original DAC work, the method is off-policy but it is compared with other on-policy algorithms, which is unfair. We suspected that DAC may be sample inefficient as compared to GAIL because DAC also has to match the state occupancy of the terminal state in addition to the existing states in the environment. For our experiments, we implement an on-policy version of DAC and observe that the method indeed is less sample efficient than GAIL.

1.3.5 Towards an unbiased reward function

We introduce the notation of ”neutral reward functions” to denote reward functions that are real valued. For example, the reward function $R(D) = \log(D) - \log(1-D)$ has the range $(-\infty, \infty)$ for $D \in (0, 1)$ and is real-valued. Neutral reward functions can be unbiased because they can penalize looping behavior with negative rewards, effectively “cancelling” the positive reward that they acquired from previous loops (assuming that the expert doesn't loop itself). The reward function can also potentially overcome 'termination bias' because the agent can collect as many positive rewards as possible by following the expert trajectories. To give an intuition of these claims, we show a similar theoretical sketch as before.

Consider an oracle discriminator that gives reward R when the agent takes the expert action a to a state s, and gives a negative reward $-R$ otherwise (since the neutral reward function is a symmetric function $R(s,a) = \log(D(s,a)) - \log(1 - D(s,a))$). To ensure the agent learns to perform the task, the discriminator should assign the highest rewards to the expert trajectories. Consider an expert trajectory of length p. Similar to the previous example, assume that each state has only one expert action. The reward of the expert trajectory is:

$$R_E = R + \gamma R + \ldots \gamma^{(p-1)} R = \frac{1-\gamma^p}{1-\gamma} R \tag{1.6}$$

The maximum reward for any other trajectory can be obtained by mimicking the expert till timestep $p-1$ to get positive rewards and do a non-expert action (to prevent the trajectory from becoming an expert trajectory). To recover from the negative reward just incurred, the agent performs an expert action. After this, it cannot perform an expert action again because that will complete the expert trajectory and the episode will terminate. So, if it wants to loop it will perform a non-expert action followed by an expert action and continue doing so. Hence the reward for this trajectory *loop* is:

$$\begin{aligned} R_{loop} &= R + \gamma R + \ldots \gamma^{(p-2)} R - \gamma^{(p-1)} R + \gamma^p R - \gamma^{p+1} R \ldots \qquad (1.7)\\ &= \frac{1-\gamma^{(p-1)}}{1-\gamma} R - \frac{\gamma^{(p-1)}}{1+\gamma} R \qquad (1.8) \end{aligned}$$

For an agent to prefer surviving rather than completing the expert trajectory, we need to have $R_{loop} \geq R_E$. This implies:

$$\begin{aligned} \frac{1-\gamma^{(p-1)}}{1-\gamma} R - \frac{\gamma^{(p-1)}}{1+\gamma} R &\geq \frac{1-\gamma^p}{1-\gamma} R \qquad (1.9)\\ \implies \frac{\gamma^{(p-1)} - \gamma^p}{1-\gamma} \leq -\frac{\gamma^{(p-1)}}{1+\gamma} &\implies 1 \leq \frac{-1}{1+\gamma} \qquad (1.10) \end{aligned}$$

Which is impossible for a discounting factor $\gamma \in [0,1]$. Hence, R_{loop} will always be lesser than R_E for an oracle discriminator, and the agent will always prefer following the expert trajectory.

A similar argument can be made for neutral rewards overcoming termination bias. Consider the previous setup, but the agent tries to terminate the episode before completing the task. The expert reward is already in Equation 1.6. To exhibit termination bias, the agent has to terminate the episode in at most p steps, and the last action should be a non-expert action (if its an expert action, then the agent is exactly imitating the expert). Hence, the maximum reward the agent can accrue is :

$$R_{term} = R + \gamma R + \dots \gamma^{(p-2)} R - \gamma^{(p-1)} R \tag{1.11}$$

which is the same as R_E for the first $p-1$ terms but the last term is a negative term whereas the last term of R_E is a positive term. Since $R_E \geq R_{term}$, the agent will prefer the expert trajectory over some other shorter non-expert path. Hence, the neutral reward overcomes termination bias.

Since the proof only requires that the reward function $R(s,a) \in [-R, R]$ with no constraints on R, we show the different choices of $R(s,a)$ like $R(s,a) = \log(D(s,a)) - \log(1 - D(s,a))$, $R(s,a) = D(s,a) - 0.5$ should achieve similar results in terms of mitigating reward bias.

1.3.6 Choice of Environment

To observe the limitations of the current methods used in adversarial imitation learning, we propose to use task based environments. However, task based environments with single terminal states will not be able to test the bias of the reward functions due to termination conditions. Hence, we propose to use environments from the Gym Minigrid package [14] with single and multiple termination conditions. Additionally, we also use the LunarLander environment since it is also a task based environment with multiple terminal states. LunarLander also provides a continuous state space representation unlike Minigrid environments.

In Minigrid, we disable the *Done* action to examine the bias of the agent only due to environmental termination conditions. To verify the effects of reward bias in a single termination condition, we choose the *Empty* and *DoorKey* environments.

1. In the *Empty* environment, the agent starts at a random location in the grid and the episode terminates when the agent steps on the goal location (in green).
2. In the *DoorKey* environment, the agent starts in a room and has to pick up a key to open a locked door. The agent has to reach the goal on the other side of the locked door.

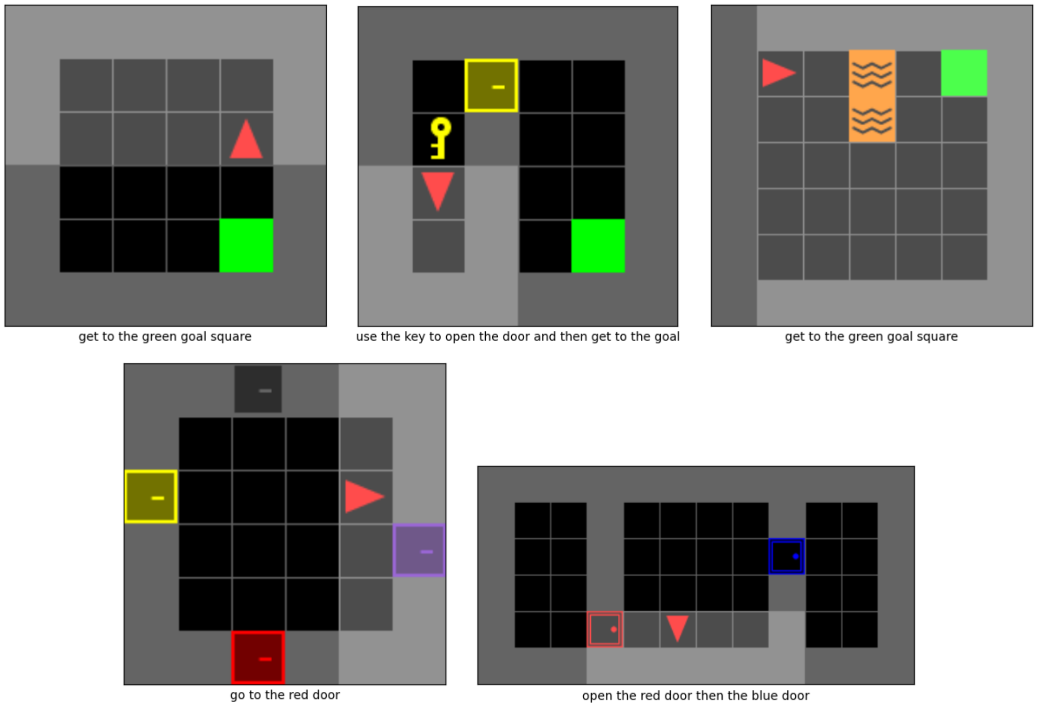
get to the green goal square
use the key to open the door and then get to the goal
get to the green goal square
go to the red door
open the red door then the blue door

Figure 1.2: Environments used in the work. The top-left and top-middle environments terminate only on reaching the goal (green). The other environments can terminate through other conditions as well (mentioned in the work). However, expert trajectories consist of goal-directed behaviors only.

Both these environments have only one termination condition - the agent has to reach the goal location. Methods like Discriminator-Actor-Critic [31] can learn a non-zero reward for the terminal condition which overcomes the survival bias of the positive reward function. GAIL with a negative reward can also learn to solve these tasks since they want to accumulate the maximum reward which would require termination of the episode as soon as possible.

However, both of these methods can suffer from the same problem - multiple termination conditions. The negative reward will have a bias towards trajectories that terminate early to accumulate fewer negative rewards. The DAC baseline will not differentiate between different termination conditions (termination due to goal completion v/s termination due to other conditions) since the terminal state can potentially provide a large cumulative reward. To test this, we use three other environments within Gym Minigrid and the LunarLandar environment.

1. *RedBlueDoors*: In this environment, the task is to open a red door, followed by opening the blue door. The episode terminates when the blue door is opened regardless of whether the red door was opened or not.
2. *GoToDoor*: This environment is similar to the *RedBlueDoors* environment but there are 4 doors of different colors, and the episode terminates when any door is opened. However, the task is considered to be completed only if the red door is opened.
3. *DistShiftv0*: In this environment, the agent has to cross a room with lava near the walls of the room (refer to Figure 1.2. The task is to reach the goal location and avoid the lava. The episode ends when the agent touches the lava or goal.
4. *LunarLandar*: In this environment, the agent has to place a lander on the landing pad. The episode finishes if the lander crashes or comes to rest, and the agent gets a positive reward if it lands correctly, in addition to small positive rewards for ground contact. The agent gets negative rewards for crashing or firing the main engine.

These environments have termination conditions which will affect the variants of GAIL depending on the reward bias that they may have. We hypothesize that the inability of DAC to differentiate between terminal states would lead to suboptimal performance in each of these environments.

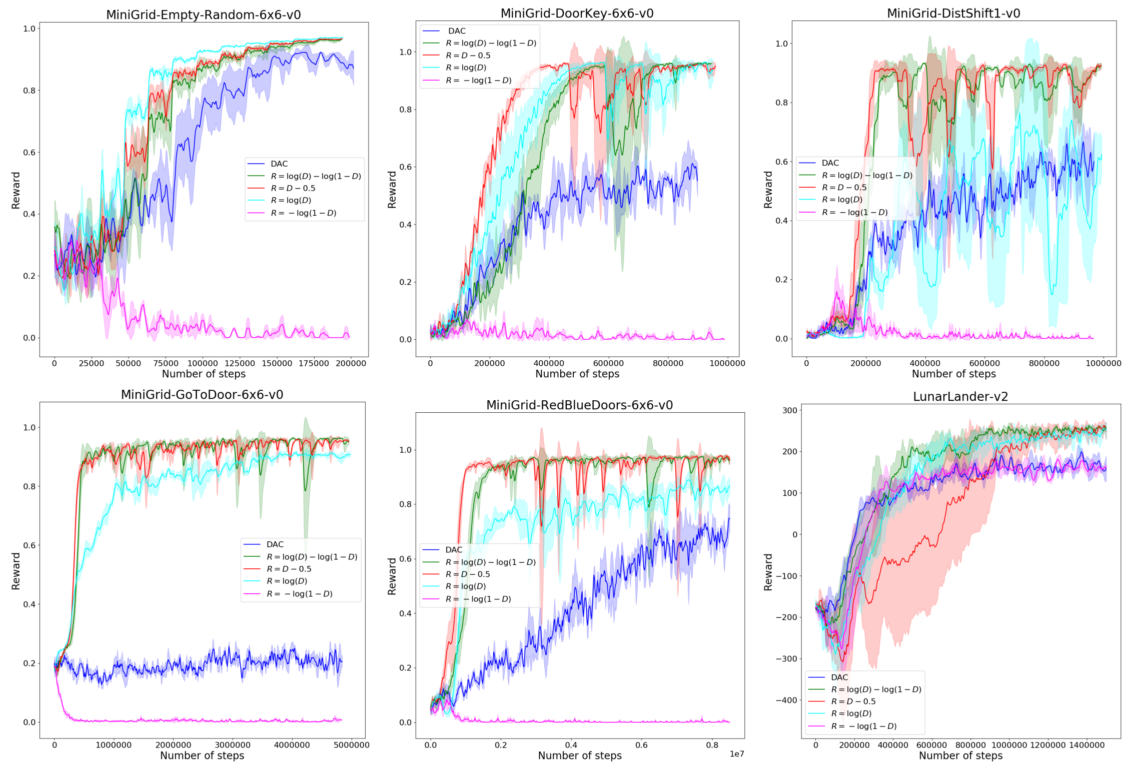

Figure 1.3: Comparison of performance of different GAIL variants. On-policy DAC is slow in terms of converging to the optimal policy, possibly due to unstable learning of the terminal state reward. In environments with non-goal terminal states, DAC and GAIL with negative rewards converge to suboptimal policies due to their respective biases.

1.4 Experiments

Our expert is a trained PPO policy. We use 1000 rollouts from the expert policy to train our imitation learning agents for all our experiments. Each rollout has ∼10 state-action pairs, so the total number of state action pairs are comparable to [13] and [27]. The locations and colors of the objects are also perturbed across episodes, so the agent has to generalize to the concept of objects rather than memorizing the goal locations. For LunarLander, we use 100 expert trajectories. We use PPO [58] for on-policy optimisation of the GAIL reward function for all experiments. In each case, the policy network takes state as input and returns probability of an action given a state. The policy network head is a 2-layered MLP with Leaky ReLU activations. The value head is identical to the policy head. We average all our results across 3 random seeds. The objective of our first experiment is to exhibit the implicit bias in the positive and negative reward functions. We compare the environment rewards achieved by the agents and the success rate for the 6 environments that we had mentioned earlier. For our second experiment, we compare the environment rewards and success rate of (i) DAC [31] (ii) Positive Reward (iii) Negative Reward (iv) Neutral Reward $R(s,a) = log(D(s,a) - log(1 - D(s,a))$ (v) Neutral Reward $R(s,a) = D(s,a) - 0.5$ in the six environments.

1.5 Results

Figure 1.1 shows reward curves for positive and negative rewards with and without learning the discriminator. The first two environments, *Empty* and *Doorkey* are able to attain expert like performance with negative rewards, which was expected as these are task based environments with single termination state. GAIL with positive reward is not able to learn the expert policy due to its survival bias (first two plots of the Figure 1.1). In case of *Empty*, the agent is able to learn the task even without training the discriminator which shows that the negative reward function has a termination bias strong enough for task based environments. Similarly, for the *DoorKey* environment, GAIL with negative reward starts to learn the task better than positive rewards even when the discriminator is not trained. However, in last three plots of Figure 1.1, where environments with multiple terminal states are used, termination bias is not enough to learn the task. Positive rewards fall into the survival bias and don't learn the task. An interesting observation is that the success rate of *GoToDoor* with negative reward and no trajectories is 0.25 as shown in Figure 1.1. The termination bias makes the agent reach out to the nearest door to end the episode, which has a 25% chance of being the red door. In Figure 1.3, we

-1cm

	Empty	**DoorKey**	**GoToDoor**	**RedBlueDoors**	**Distshift1**	**LunarLandar**
$R = -\log(1-D)$	0.03 ± 0.15	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	155.81 ± 63.92
$R = -\log(0.5)$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	-15087.98 ± 2657.88
$R = \log(D)$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.93 ± 0.24	0.83 ± 0.37	0.85 ± 0.35	239.93 ± 79.84
$R = \log(0.5)$	$\mathbf{1.00 \pm 0.00}$	0.25 ± 0.43	0.38 ± 0.48	0.00 ± 0.00	0.65 ± 0.47	-449.43 ± 76.86
DAC	$\mathbf{1.00 \pm 0.00}$	0.83 ± 0.37	0.26 ± 0.44	0.70 ± 0.45	0.76 ± 0.42	156.20 ± 95.53
$R = \log(D) - \log(1-D)$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.97 \pm 0.15}$	0.96 ± 0.19	0.97 ± 0.15	$\mathbf{256.20 \pm 55.60}$
$R = D - 0.5$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.95 ± 0.15	$\mathbf{0.98 \pm 0.14}$	$\mathbf{0.99 \pm 0.10}$	248.46 ± 73.18

Table 1.1: Success rate of all learnt policies across 3 seeds. For each seed, 80 trajectories were recorded and the mean and standard deviation is calculated across all trajectories.

compare the performance of DAC, neutral rewards, negative and positive rewards. The success rate of the different methods is also summarized in Table 2.1. For *Empty* and *DoorKey* all methods except GAIL with positive rewards are able to learn the task. However, DAC is slower than other GAIL methods, and we hypothesize this is because DAC has to learn the reward for the terminal state as well. For the other three environments, termination bias hinders learning with negative rewards and DAC suffers in performance due to lack of distinction between terminal states. For *GoToDoor*, *DistShift* and *RedBlueDoors*, DAC is not able to achieve high success rate showing its limitations in task based scenarios with multiple terminal states. Success rate of neutral reward is the highest and better than the negative reward showing its robustness to termination bias. Similarly, for other the other two task based environments with multiple terminal states, neutral reward significantly outperforms both DAC and negative reward. GAIL with neutral rewards is able to outperform all other methods, hence, is able to overcome both survival and termination bias. In case of LunarLander, the positive reward leads to a score below 200, which is the threshold for an episode to be considered "successful". The negative reward function mitigates this bias but still underperforms because it tries to complete the episode faster to collect lesser negative rewards. Neutral reward functions mitigate both kinds of bias and result in optimal task reward by imitating the expert. Although the final score is not very low for all methods, an analysis of the average episode length shows that agents with positive reward functions deliberately lengthen the duration of the task completion to collect more positive rewards. The agent with negative reward has a slightly lower average episode length showing a tendency to complete the episode quickly. We include this analysis in the Supplementary Material.

1.5.1 Effect of different reward functions on episode length

In Figure 1.3 we analysed the effect of different GAIL rewards on the agent's ability to learn the task. Here, we confirm the effects of our method by checking the average episode lengths of the methods (in Figure 1.4). In all environments, GAIL with positive rewards fails to learn the task, and the episode lengths always reach the time-limit, therefore confirming that survival bias in GAIL is indeed a practical problem. In environments like *Empty* and *DoorKey*, negative rewards are better since there is a single termination condition corresponding to task completion. We see that this method indeed performs better than GAIL with positive rewards. This reward is closely followed by neutral rewards which also obtain a similar curve for average episode length. However, in the other environments with multiple terminal conditions, we can see a clear difference between negative and neutral rewards. The difference is clearly visible in *GoToDoor* where there is a non-negligible gap between negative and neutral rewards. Although the average episode length for negative rewards is lower, the success rate is also lower, which confirms that negative rewards suffer from a termination bias problem. Neutral rewards take slightly more time on average to end an episode, but that is because they actually complete the task and do not terminate an episode early. In LunarLander, the difference in final rewards does not seem to be significantly different. However, the average episode lengths tell a different story. In case of positive rewards, the average episode length goes upto 900 timesteps, which is far more than the required timesteps taken to land the launcher. GAIL with negative reward seems to have a lower episode length throughout training, which can be attributed to its termination bias. However, neutral rewards have similar average episode lengths at convergence, and perform better than GAIL with negative reward. This shows that finishing an episode quickly does not necessarily translate to better

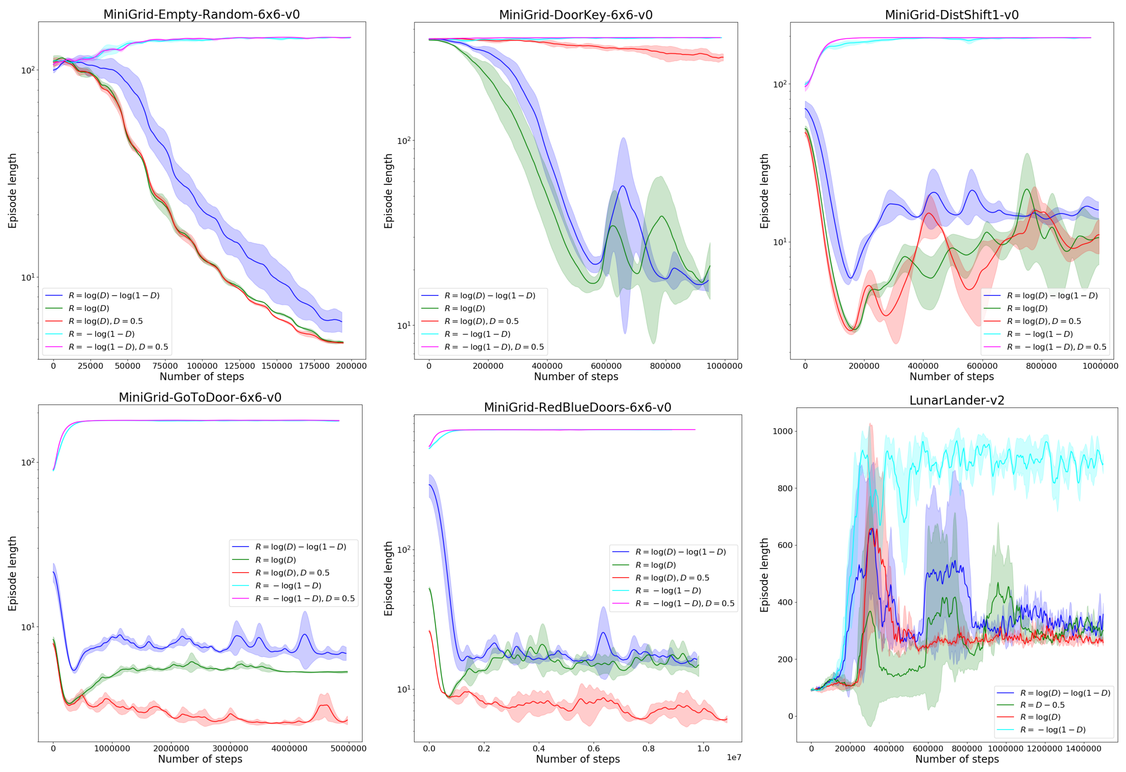

Figure 1.4: Comparison of average episode length for different GAIL reward functions.

task performance. GAIL with neutral rewards seem to balance this trade-off most optimally across all our experiments.

1.6 Conclusion

In this work, we address the problem of reward bias in adversarial imitation learning. We explore two types of reward biases - survival bias and termination bias, and how different reward functions lead to these biases in the agent. Positive reward functions encourage survival bias, and negative reward functions encourage termination bias, and both these biases may hinder learning in a task-based environment. We show that real-valued reward functions are unbiased and can learn to overcome both survival and termination biases. Experiments show that this simple change of reward function enables the agent to imitate the expert on tasks with single and multiple termination conditions.

Chapter 2

Adaptative Agent Architecture for Human-Agent Teaming

2.1 Introduction

Multi-agent systems have recently seen tremendous progress in teams of purely artificial agents, especially in computer games [24, 46, 67]. However, many real-world scenarios like autonomous driving [19, 52], assisted robots [4, 39], and Unmanned Aerial System [16, 44] do not guarantee teams of homogeneous robots with shared information - more often, it involves interaction with different kinds of humans who may have varying and unknown intents and beliefs. Understanding these intents and beliefs is crucial for robots to interact with humans effectively in this scenario. **Human-agent teaming** (HAT) [11, 56], an emerging form of human-agent systems, requires teamwork to be a set of interrelated reasoning, actions and behaviors of team members that combine to fulfill team objectives [45, 54, 55]. In this work, we focus on the setting of two-player human-agent teaming in a computer game, where the agent should cooperate with the human in real-time to achieve a common goal on one task. The human playing that role may be any person with any policy at any time, and potentially not be an expert in the task at hand.

One of the fundamental challenges for an artificial agent to work with a human, instead of simply another artificial agent, is that humans may have complex or unpredictable behavioral patterns and intent [10, 23]. In particular, they may misuse or disuse the multi-agent system based on their perception, attitude and trust towards the system [47]. This difference becomes very critical in scenarios where an agent interacts with a diverse population of human players, each of which might have different intents, beliefs, and skills (ranging from novices to experts) [36]. To succeed, cooperative agents must be able to infer human intent or policy to inform their action accordingly.

Capabilities of adapting to humans are essential for a human-agent team to safely deploy and collaborate in a real-time environment [9]. Real-time adaptation is critical in practical deployments where robots are not operating unilaterally in a controlled environment, such as urban driving environments for autonomous vehicles [19]. The ability to respond to real-time observations improves an agent's ability to perform in the face of various kinds of team structures or situations. Accomplishing real-time agent adaptation requires that agents are able to capture the semantics of the observed human behavior, which is likely volatile and noisy, and then infer a best response accordingly. The challenge of capturing human behavior is further increased since the agent only observes a small snapshot of recent human actions, among players with varying play styles or skill levels. Finally, we note that humans may adjust their behavior in response to a changing agent policy, which can make stable adaptation difficult to achieve [19, 26, 53]. Real-time environments like computer games also require agents to perform both sufficiently fast estimation of the teammate's policy, as well as planning, while ensuring flexibility for unexpected strategic game states [68].

Past research on real-time adaptation in HAT can be divided into two forms of adaptive agent training. In the first *human-model-free* form: the agent does not build a model of human policy, but instead infers human types from the match between current observations and an exemplar and then take corresponding best actions. This setting is adopted by our approach and can be found in the psychology literature [32, 33, 34]. The second form widely adopted in robotics research *human-model-based*: first trains a model of the human to learn humans' policies or intent, then integrates the human model into the environment, and finally trains the agent upon the integrated environment. This setting requires much more computational resources than *human-model-free* form to learn the human model and deploy the agent in real-time, and inevitably imposes heavy assumptions on human policies [19, 52, 70] like: optimal in some unknown reward function, single-type or consistent among different humans, and time-invariant for one human, etc. However these assumptions deviate from real-world human policies, especially coming from a diverse population in different skill levels and intent on a relatively hard task. On the contrary, *human-model-free* setting imposes minimal assumptions on human policies and can be deployed in real-time teaming efficiently.

In this work, we propose a human-model-free adaptive agent architecture based on a pre-trained static agent library. The adaptive agent aims to perform well in a nontrivial real-time strategic game, Team Space Fortress (TSF) [3]. TSF is a two-player cooperative computer game where the players control spaceships to destroy the fortress. TSF presents a promising arena to test intelligent agents in teams since it involves heterogeneous team members (bait and shooter) with adversary (fortress), and it has sparse rewards which makes model training even more difficult. TSF is a nontrivial testbed to solve as it requires some real-time cooperation strategy for the two players without communication and control skills for human players. Before constructing exemplar policies, we first evaluate the nature of this testbed through previous research in *human-human* teams. The results [40] show that different human-human pairs demonstrate significantly diverse performance and the team performance was affected by both individual level factors such as skill levels and team level factors such as team synchronization and adaptation.

The diverse team performance and complicated team dynamics in *human-human* teams inspired us to build a real-time adaptive agent to cooperate with any human player in *human-agent* teams. The methodology of our real-time adaptive agent is quite straightforward. First, we design a diverse policy library of rule-based and reinforcement learning (RL) agents that can perform reasonably well in TSF when paired with each other, i.e. *agent-agent* teams. We record the self-play performance of each pair in advance. Second, we propose a novel similarity metric between any human policy and each policy in the library from observed human behavior, namely cross-entropy method (CEM) adapted from behavior cloning [6]. The adaptive agent uses the similarity metric to find the most similar policy in the exemplar policies library to the the current human trajectory. After this, the adaptive agent switches its policy to the best complementary policy to the predicted human policy in real-time. Using this adaptive strategy, it is expected to outperform any static policy from the library. Our approach is directly built upon single-agent models, thus can generalize to any off-the-shelf reinforcement learning/imitation learning algorithms.

We evaluated our approach online by having human players play against both agents with exemplar static policy and adaptive policy. These players were sourced through Amazon's Mechanical Turk (MTurk)[1] program and played TSF through their internet browsers. Each human player was assigned one role in TSF and played with all the selected agents for several trials, but was not told which agents they were playing with and was rotated through random sequences of the agents to ensure agent anonymity and reduce learning effect.

Based on the collected game data from these human-agent teams, we are interested in the three key questions: (1)How are human players' policies compared to agent policies in our library? (2) Is our adaptive agent architecture capable of identifying human policy and predicting team performance for human-agent teams? (3) Do our adaptive agents perform better than static policy agents in human-agent teams? We answer these questions in the experimental section.

2.2 Related Work

In the multi-agent system domain, researchers have been focusing on how autonomous agents model other agents in order to better cooperate with each other in teams, which is termed as *human-model-based* methods when applied to human-agent system in the introduction of this work. Representative work includes *ad-hoc teamwork* that the agent is able to use prior knowledge about other teammates to cooperate with new unknown teammates [5, 8].

This is a reasonable number of work in human-robot interaction that attempts to infer human intent from observed behaviors using inverse planing or inverse reinforcement learning [7, 19, 48, 51, 52]. However, these work impose ideal assumptions on human policy, e.g. optimal under some unknown reward, consistent through time, and with unique type among humans, which does not hold in many complicated real-world applications, where the human-agent systems are required to generalize to various kinds of team scenarios.

[1]https://www.mturk.com/

In human-agent teaming, past research [12, 18, 25, 37, 66] has established a variety of protocols within small teams. However, these approaches often rely on some degree of explicit communication on humans' observation or intent.

The alternative setting of agent design in human-agent system is *human-model-free*, rarely discussed in the robotics literature. Some psychology literature in this setting learns to infer human intent from retrospective teammate reports, where software analyzes historical observations of humans to inform behavior in the present [32, 33, 34]. These historical behaviors may fail to capture potential changes in teammate policies a real-time environment and limit the ability of software to best adapt to a situation.

Our approach opens the door of *human-model-free* setting in human-agent system for robotics literature, significantly different from previous *human-model-based* methods. We make least assumptions on human, and use proposed architecture to realize adaptation, which involves similarity metric to infer human policy. The least assumptions and straightforward architecture enable our approach to deploy in a real-time human-agent environment with various kinds of human players.

2.3 Team Space Fortress

We have adapted Space Fortress [43], a game which has been used extensively for psychological research, for teams. Team Space Fortress (TSF) is a cooperative computer game where two players control their spaceships to destroy a fortress in a low friction 2D environment. The player can be either human or (artificial) agent, thus there are three possible combinations in teams: human-human, human-agent, and agent-agent.

A sample screen from the game is shown in Fig. 2.1. At the center of the stage lies a rotating fortress. The fortress and two spaceships can all fire missiles towards each other at a range. The first spaceship entering the hexagon area will be locked and shot by the fortress. However, the fortress becomes vulnerable when it is firing. Players die immediately whenever they hit any obstacles (e.g. boundaries, missiles, the fortress). The game resets every time either fortress or both players are killed. Once the fortress has been killed, both players must leave the activation region (outer pink boundaries) before the fortress respawns.

The team performance is measured by the number of fortresses that players kill.

The action space is 3-dimensional discrete space, including TURN (turn left, right, or no turn), THRUST (accelerate the speed or not), and FIRE (emit one missile or not). The frame per second (FPS) is 30, thus in a 1-minute game, there are around 1800 frames.

In order to test a common instance of teamwork, players were instructed in a common strategy and assigned roles of either *bait* or *shooter*. The *bait* tries to attract the fortress's attention by entering the inner hexagon where it is vulnerable to the fortress. When the fortress attempts to shoot at the bait, its shield lifts making it vulnerable. The other player in the role of *shooter* can now shoot at the fortress and destroy it.

There are some difference in observations and actions between human players and agent players. Human players observe the game screen (RGB image) at each frame, then hit or release the keys on keyboard to take actions. Agent players instead observe an array composed of the states (position, velocity, angle) of all the entities including the fortress and two players, and communicate their actions directly through the game engine.

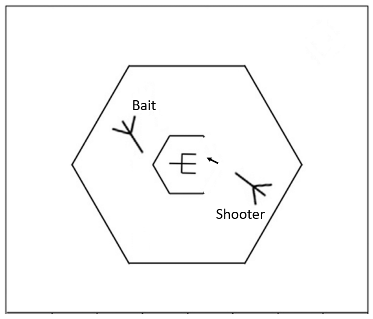

Figure 2.1: Sample TSF game screen (line drawing version, original screen is in black background). Spaceships are labeled as shooter and bait. Entity at center is the rotating fortress with the boarder around it as the shield. Activation region is the hexagon area around players' spaceships. Black arrow is a projectile emitted from the shooter towards the fortress. All the entities are within the rectangle map borders.

2.4 Adaptive Agent Architecture

In this section, we formulate our method for an adaptive agent architecture. First we introduce the exemplar policies library pre-trained by reinforcement learning or designed by rules for TSF. This library will be used as a standard baseline to identify human policies. Next, we introduce the similarity metric adopted in the architecture (i.e. cross-entropy metric) that measures the distance between human trajectory and exemplar policies in the library. Finally, we define the adaptive agent architecture given the estimated human policy according to the similarity metric.

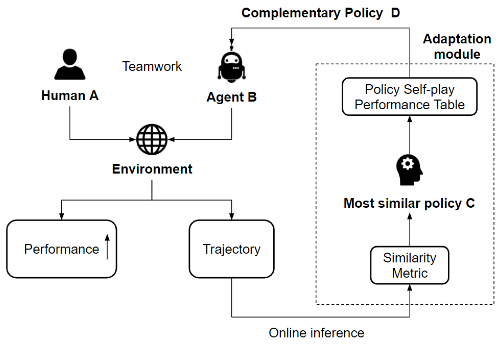

Figure 2.2: The flowchart of the proposed adaptive agent architecture. The adaptation module (in dotted boarder) takes the input of the trajectory at current timestamp, and then assigns the adaptive agent with new policy at next timestamp. The adaption procedure can be deployed in real-time (online).

2.4.1 Exemplar Policies Library

The exemplar policies library $\mathcal{L} = \mathcal{L}_{\cup}\mathcal{L}_{\S}$ consists of two sets of policies in bait () and shooter (§) roles, $\mathcal{L}$ and $\mathcal{L}_{\S}$ respectively. Both bait policies and shooter policies are trained using a combination of RL and rule based behavior.

These exemplar policies can be divided into several main types: baits can be divided into three types (B1-B3, B4-B7, B8-B9) and shooters can be divided into two types (S1-S3, S4-S7). Below are the technical details of each type.

Bait policy library $\mathcal{L}$

To make these different bait policies diverse, we train them using different reward functions, inspired from human-human experiments where there were multiple ways to achieve good performance. The reward functions attempt to encode the desirable behavior of a bait agent. The bait agent is then trained using an RL algorithm to achieve an optimal behavior with respect to the given reward function. The bait library $\mathcal{L}$ are composed of 9 bait policies. The goal for bait is to keep alive inside the activation region to make the fortress vulnerable from behind so that the shooter can grasp the opportunity to destroy the fortress from behind. In general, the bait has two *conflicting* objectives which it tries to balance. If the bait is inside the activation more time, it is more vulnerable and prone to getting killed by the fortress. However, bait's presence inside the activation region gives an opportunity for the shooter to attack the fortress from behind. Different bait agents try to balance these two conflicting objectives in different ways.

B1-B3 type policies are trained by A2C algorithm [30] to learn TURN action and use rules on THRUST action. The reward function of TURN learning is binary, encouraging the baits to stay inside the activation region. For the observation space of the bait policy, we convert the original Cartesian coordinate system to a new one, where the new origin is still at the fortress while the new positive Y-axis goes through the bait, which is shown to ease the training in RL for TSF. Then, we use the converted coordinates of bait position and two nearest missile positions to train the agent. The intuition is that bait is sensitive to the nearest shells to keep itself alive. The rule in THRUST action limits the maximum speed of the bait agents. By tuning the threshold in speed, we have policy B1-B3. B4-B7 policies are trained by RL in both TURN and THRUST actions. They share same reward function as B1-B3 using the same transformation in coordinate system. By using different RL algorithms from A2C [30], PPO [59], to TRPO [57], and different observation space (whether to perceive the shooter's position and velocity), we have policies B4-B7.

B8-B9 belong to the another set of policy using a different reward structure, learning TURN and THRUST by PPO algorithm [59]. B8 and B9 are designed by aggressive and defensive objectives, respectively. The reward structure used to train these 2 agents is composed of three parts: (1) "border reward" to encourage the agents to keep far away from the fortress, (2) "bearing reward" to encourage the agents to align itself directly towards the fortress when inside the activation region, (3) "death penalty" to discourage the agents from being killed by the fortress or hitting the border. Border reward encourages risk-averse behavior, while bearing reward risk-seeking behavior, and by controlling the coefficients among these three parts, we have agents B8-B9.

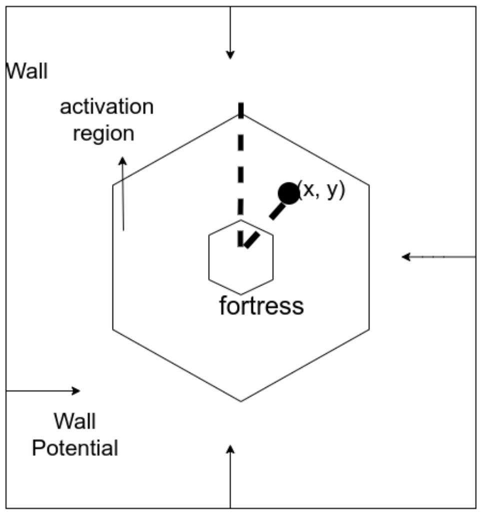

Figure 2.3: TSF Environment

More details on B8 and B9

This bait agent was trained using PPO[60] algorithm. The reward function used to train this bait is as follows.

The reward consists of three components.

- Wall Reward: Each wall applies a repulsive reward on the bait agent that is proportional to e^{-d} where d is the distance of the bait from the wall. So, for the bait coordinates (x, y), the reward is:

$$r_{wall} = e^{-(x_{wall1}-x)} + e^{-(x_{wall2}-x)} + e^{-(y_{wall3}-y)} + e^{-(y_{wall4}-y)}$$

  This is the total wall reward for the 2 horizontal and the 2 vertical walls.

- Bearing Reward: The bait, when inside the activation region gets rewarded for aligning itself directly towards the fortress.

$$r_{bearing} = cos(\theta)$$

  Here θ is the angle between the fortress head and the bait.

- Environment rewards: These include negative rewards for getting killed after hitting a wall, a shell or a the shield

So, the total reward is:

$$r_{bait} = K r_{wall} + K' r_{bearing} + \lambda(r_{hit-shell} + r_{hit-shield} + r_{hit-wall})$$

where K, K', λ is the weight.

Aggressive Bait - B8

For training this bait the wall repulsive reward(parameterised by K) was set high. Also, the bearing rewards(parameterised by K') was also set to a higher value and the hit shell and hit shield penalities were kept to low so as to encourage risky behavior.

Defensive Bait - B9

For training this bait the wall repulsive reward(parameterised by K) was set lower. Also, the bearing rewards(parameterised by K') was also set to a lower value and the hit shell and hit shield penalities were kept to high so as to discorage risky behavior.

Shooter policy library $\mathcal{L}_{\S}$

The shooter policy library $\mathcal{L}_{\S}$ are composed of 7 shooter policies, with 4 of them mirror shooters that are purely rule-based, and 3 of them RL shooters that learn the TURN action by RL.

The mirror shooters are based on the prior knowledge of TSF game that a good shooter should have an *opposite* position against the Bait, which was observed in many successful human-human teams. Thus the mirror shooter tries to keep at opposite position to the current position of the bait (termed as *target position*) until it finds itself having a good chance to fire to destroy the fortress. By controlling the threshold of distance to the target position, we have agents S4-7.

The RL shooters' reward function takes the same team strategy of opposite positions, trained by DDQN [65] on TURN action. Specifically, the reward is designed to encourage the shooter to keep close to the outside the activation region when bait does not enter the region, and keep at the rear of the fortress when bait enters the region. S1-3 are different in max speed.

Self-play performance

We evaluate the performance of each shooter-bait pairs in the exemplar policy library by self-play in TSF environment, and record the results in self-play performance table $\mathcal{P}$ in advance. The table $\mathcal{P}$ has rows with the number of bait policies in $\mathcal{L}$ and columns with the number of shooter policies in $\mathcal{L}_{\S}$, with each entry the average performance of the bait-shooter pair. When applied to our policy library, table $\mathcal{P}$ is showed in Table 2.1.

On average, teams that consist of two static agents show significantly better performance (6.03) than human-human teams (2.60) reported in previous research [40]. This indicates that most of our agent pairs, including both RL-based and rule-based agents, have a super-human performance in TSF which benefits from the design of reward function and rules.

Similar to human-human teams, agent-agent teams also show complementary policy pairs that work extremely well with each other. An example would be S4-S7 (mirror shooters) who yield a dominant performance when pairing with most of the baits except for B8 and B9. While for specific bait policies such as B8 and B9, the best teammate would be S2 or S3 (RL shooters) in stead of the more "optimal" S4-S7. We could tell from the self-play table that the space of reasonable policies in TSF game is indeed diverse, and there are more than one path towards good team dynamics and team performance. This confirms again the necessity of introducing real-time adaptive agents in human-agent teams. Thus we build the adaptive agent framework based upon this self-play table in the next subsections.

2.4.2 Similarity Metrics

Cross-Entropy Metric

Now we introduce the **cross-entropy metric** (CEM) as the similarity metric used in this architecture. Cross-entropy, well-known in information theory, can measure the (negative) distance between two policies π_1, π_2:

$$\mathrm{CEM}(\pi_1, \pi_2) = E_{s,a\sim\pi_1} \log \pi_2(a|s) \tag{2.1}$$

	S1	S2	S3	**S4**	**S5**	**S6**	**S7**
B1	5.1	5.7	5.0	5.1	4.9	4.5	3.9
B2	6.5	7.0	6.0	7.6	7.6	7.5	6.8
B3	5.9	6.7	5.8	8.0	8.1	8.2	7.9
B4	6.4	7.1	5.9	7.5	7.6	7.3	7.3
B5	6.3	7.1	6.2	6.8	6.8	6.4	6.2
B6	6.2	7.1	6.1	7.8	7.8	7.4	7.0
B7	6.2	7.0	6.2	7.8	7.8	8.0	7.8
B8	4.3	5.3	**5.4**	3.1	2.8	3.0	3.0
B9	4.9	5.7	5.5	2.3	2.1	2.1	1.8

Table 2.1: Self-play agent performance table $\mathcal{P}$. Each row is for one bait agent named Bi in $\mathcal{L}$ (i=1 to 9), and each column is for one shooter agent named Sj in $\mathcal{L}_{\S}$ (j=1 to 7). Each entry is computed by per-minute team performance (number of fortress kills) of the corresponding pair. We segment the tables to group same type of agents, and mark the "optimal" bait and shooter agents in bold.

where $\pi_1(\cdot|s), \pi_2(\cdot|s)$ are action distributions given state s. This is actually the training objective of behavior cloning [6] to expert policy π_1, i.e., $\max_{\pi_2} \text{CEM}(\pi_1, \pi_2)$, which is to maximize the log-likelihood of expert actions in agent policy π_2 given a collection of expert state-actions. Thus the larger the $\text{CEM}(\pi_1, \pi_2)$, the more similar π_1 is to π_2.

If we know the policy π_2, and are able to obtain state-action samples from π_1, then we can estimate cross-entropy $\text{CEM}(\pi_1, \pi_2)$ by Monte Carlo sampling. That is to say, under the assumption above, policy π_1 can be unknown to us. In human-agent teaming, human policy π_H cannot be observed but the state-action pairs generated by the human policy can be easily obtained, and agent policy π_A is designed by us, as programmers, thus known to us.

Therefore, we can leverage CEM as the similarity metric: given a sliding window of frames that record the observed behavior of the human policy π_H, we can estimate the cross-entropy between a human policy π_H and any known agent policy π_A by the following formula:

$$\frac{1}{T}\sum_{t=1}^{T} \log \pi_A(a_t|s_t), \quad \text{where } (s_t, a_t)_{t=1}^{T} \sim \pi_H \tag{2.2}$$

where $(s_t, a_t)_{t=1}^{T}$ are the sequential state-action pairs from human policy, T is the

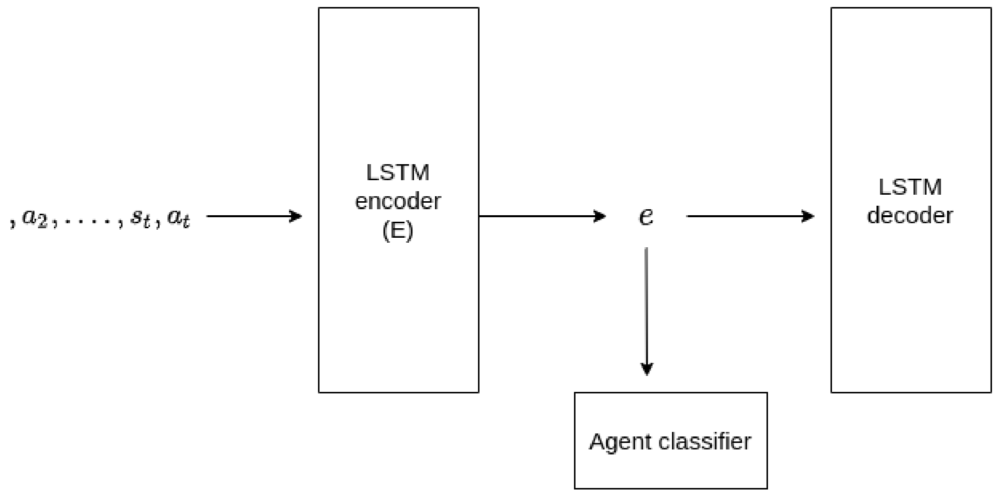

Figure 2.4: Policy Representation Architecture

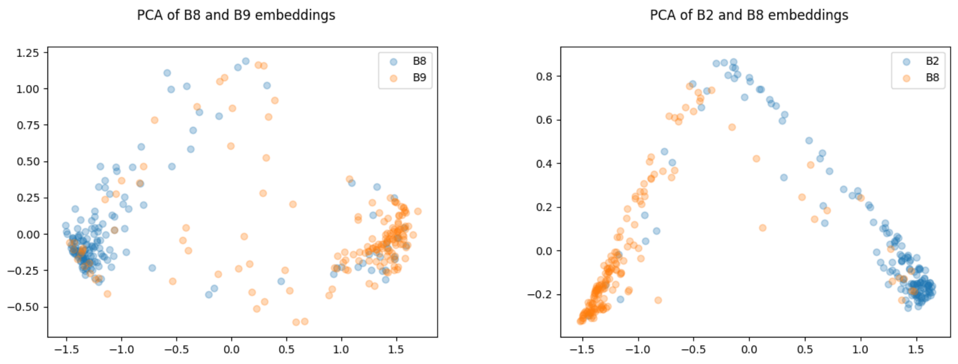

Figure 2.5: PCA of the policy representation of different bait agents

window size, which is a hyperparameter to be tuned.

An LSTM based policy representation model is trained on the agent trajectories from the agent library. The policy representation models are learnt separately for bait and the shooter. The model is an LSTM autoencoder which takes in state-action sequence of the agent as an input and learns a fixed lower dimensional embedding of a trajectory. In this way, we are able to encode a trajectory of any given length using these policy representation models. We store the embeddings for the trajectories of the agents in the agent-agent library. To make the embeddings more robust we also use an auxiliary task of agent classification. This model is able to separate out trajectories from different agent policies as shown in Fig2.5, 2.6 Further, it is also

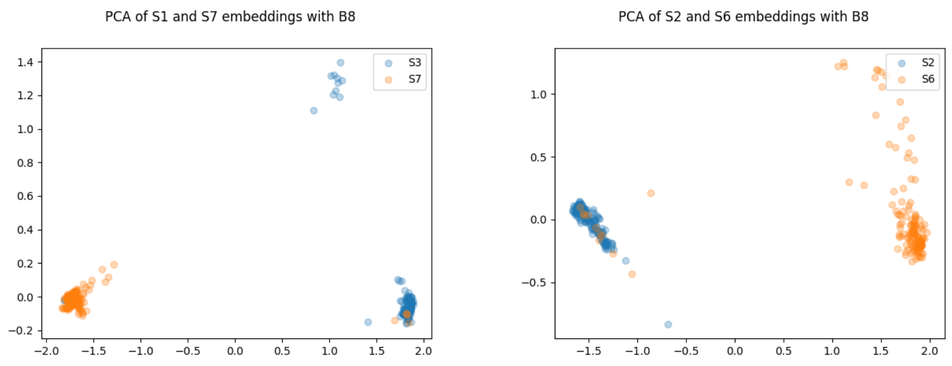

Figure 2.6: PCA of the policy representation of different shooter agents

possible to joint policy representation models of bait and shooter from this.

This model is then used to define Autoencoder Similarity Metric (ASM). ASM uses an recurrent autoencoder trained to identify an agent given the historically observed behavior from known agents in agent policy library introduced in Sec 2.4.1. By translating observed behavior into the latent space of the autoencoder, the encoder effectively generates an embedding from these observations. The embedding of the human trajectory is then compared against embeddings of different agents in the policy library. Since this model uses a recurrent network its too slow to be used for real time adaptation. However, we still use it offline to see trends human policy consistency.

2.4.3 Adaptive Agents

The prerequisite for the architecture is the exemplar policies library $\mathcal{L}$ introduced in the Sec. 2.4.1 and the self-play table $\mathcal{P}$ of the library to translate human-agent performance in the adaptation process.

Figure 2.2 shows the overall flowchart of our adaptive agent framework. When the game starts and a new human player A starts to play as one pre-specified role $R_1 \in \{, \S\}$ in TSF, the adaptive agent framework will first randomly assign a policy B from the library $\mathcal{L}_{R_2}$ in teammate role R_2 such that $\{R_1, R_2\} = \{, \S\}$, and keep track of the joint trajectories (state-action sequences) and record them into memory.

The adaptation process is as follows. As we maintain the latest human trajectories of a pre-specified window size, and we first use the data to compute the similarity by cross-entropy metric between the human trajectory and any of exemplar policies in the library $\mathcal{L}_{R_1}$ with same role. Then we figure out the most similar policy $C \in \mathcal{L}_{R_1}$ to the human trajectory, and look up the performance table $\mathcal{P}$ to find the optimal complementary policy $D \in \mathcal{L}_{R_2}$ to the predicted human policy type C. Finally, we assign the agent D as the complementary policy at next timestamp with the human player.

The adaptation process on the exemplar policies selection is based on the following assumption: if the human policy A with role R_1 is similar to one exemplar policy $C \in \mathcal{L}_{R_1}$ within some threshold, then the human policy A will have similar team performance with teammates as C, i.e., if C performs better with $D \in \mathcal{L}_{R_2}$ than $E \in \mathcal{L}_{R_2}$, so does A. This enables us to adapt the agent policy in real-time by the recent data without modeling the human policy directly.

2.5 Human-Agent Teaming Experiments

In this section, we first introduce our experiment design for human-agent teaming, then evaluate the human-agent performance when paired with static policy agents (introduced in Sec. 2.4.1) and proposed adaptive agents (introduced in Sec. 2.4.3).

By analyzing the collected human-agent data, we aim to answer the following motivated questions:

1. How are human players' policies compared to agent policies in our library?
2. Is our adaptive agent architecture capable of identifying human policies and predicting team performance for human-agent teams?
3. Do our adaptive agents perform better than static policy agents in human-agent teams?

2.5.1 Experimental Design

We recruited participants from Amazon Mechanical Turk for our human-agent experiments. They were paid USD 2 for participating in the 15-min online study. Participants were randomly assigned a role of either shooter or bait and then teamed with artificial agents in the corresponding role to play Team Space Fortress. Each participant would need to complete five sessions of data collection with three 1-min game trials in each session. Participants teamed with different agent variants between sessions in a random sequence. The five variants were selected from our static agent library $\mathcal{L}$. When selecting these designated agents, we balanced the performance in self-play table and the diversity by considering different training methods and reward functions. Specifically, we select {B3, B6, B7, B8, B9} as tested static baits and {S1, S2, S3, S4, S7} as tested static shooters. In the dataset of human and static agent teams, we got 25 valid data points from human shooters and 29 valid data from human baits.

2.5.2 Results

Policy space representation

To quantify the relationship between real human policies in the experiments and agent policies in the library, we leveraged a similarity embedding by comparing the distance between the collected human trajectories and agent policies using CEM measurement (see Sec. 2.4.2). This provides us with a high-dimensional policy space based on agent policies in our library. Specifically, CEM was employed to generate the average log-probability of state-action pairs in a human trajectory coming from a certain agent policy. We could then construct a similarity vector for each trajectory with the dimensions equal to the number of policies in the library. The value in each dimension represents the similarity distance from human trajectories to a certain agent policy. Then, we applied a principal component analysis (PCA) based on the log-probability dataset to project the high-dimensional policy space into a 2D plane for a better visualization. The two primary components left explain more than 99% of the variance.

Fig. 2.7 illustrates the human policies in static agent dataset. We could get following qualitative insights from the illustration: 1) the learnt similarity embedding separates different human policies well, 2) reinforcement learning policies are homogeneous (red nodes in the bottom-left corner) while the rule-based policies are a bit off (red nodes in the upper-left corner). 3) the distribution of human policies correlates with their team performance in that players to the left tend to have better team performance (colored nodes to the left are larger in size). Those findings align with our expectations and validate the proposed adaptive agent architecture. In the following analysis, we will quantify them based on the CEM measurement and the similarity embedding.

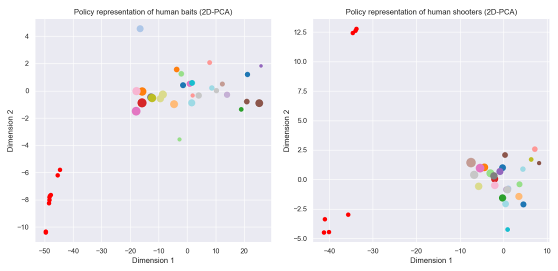

Figure 2.7: Policy representations of each human baits (left) and shooters (right) in the static agent dataset (after PCA dimension reduction). Each colored node in the figures represents the average policy of a human player, while the size of which indicates his average team performance. Red nodes are reference points of baseline agent polices.

Human policy identification

In the proposed adaptive agent architecture, our model infers human policy by classifying it as the most similar policy in the library based on CEM measurement, then assigns the agent with the corresponding complementary policy in the self-play table. One way of verifying this method is to see if human-agent teams performed better when the predicted human policy was closer to the complementary match in the self-play table $\mathcal{P}$. Assuming each human maintains a consistent policy over the course of interaction when paired with a specific teammate with static policy, we could then calculate, for each human-agent pair, the similarity between human policy and the optimal agent policy for the agent that the human was playing with.

This "similarity to optimal" quantifies the degree to which a human player is similar to the optimal policy given an agent teammate in our architecture. Correlation analysis shows that "similarity to optimal" is positively correlated with team performance in both bait ($r = 0.636, p = .0002$) and shooter ($r = 0.834, p < .0001$) groups. This result indicates that the complementary policy pairs we found in agent-agent self-play can be extended to human-agent teams, and our proposed architecture is able to accurately identify human policy types and predict team performance.

Furthermore, our model could also infer human policies in real-time. This is to say, even within the same team, humans might also take different sub-policies as their mental model of the team state evolves over time [54]. We could take the log-probabilities generated by CEM as time series data to capture the online adaptation process of humans over the course of interaction at each timestamp.

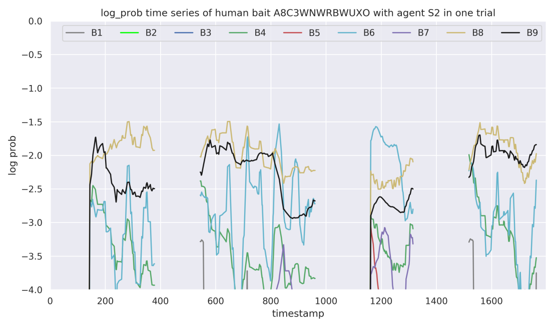

Figure 2.8: The log-probability curves of one human policy generated by CEM. Data is from one specific 1-min trial of a human bait paired with agent S2. We segment the trial into several episodes, each of which starts with the bait entering the activation region and ends with the team killing the fortress. The curves with same color represent the same agent policy for inference.

An example visualization is shown in Fig. 2.8 where curves represent the log-probabilities of each agent policy over the course of interaction. We can tell from the graph that in the segments of a specific trial, the human trajectories were inferred to reflect different policies, although the average log-probability would still be in favor of B8. Those findings motivate us to test an online adaptive agent using a sliding time window to capture the human policy shifts in real-time.

2.5.3 Trends in Performance and Similarity to Optimal Partner using ASM metric

We plot the performance and similarity to most optimal partner of human shooters and human baits against different agents from our policy library. We try to see if the performance of the team increases as the similarity to the best complimentary policy increases. However, we don't observe any consistent trends. For some agents as shown in Fig2.9, we do observe that the performance and the similarity follow a similiar trend. However, Fig2.10 it is not observed for all humans.

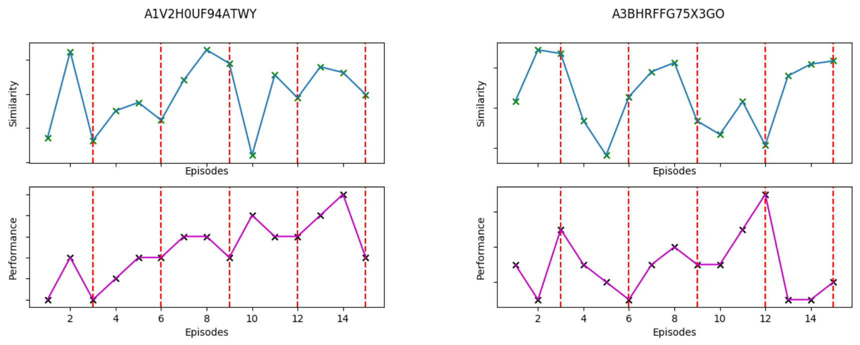

Figure 2.9: Performance of humans and simiarity to the most optimal partner

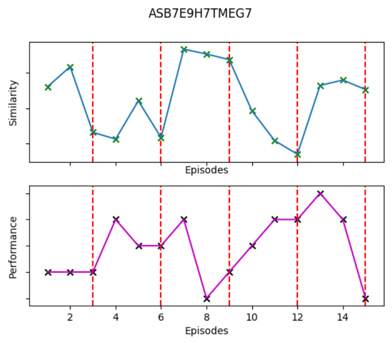

Figure 2.10: Performance of humans and simiarity to the most optimal partner

2.5.4 Pilot experiment with adaptive agents

In previous experiment and analysis, we validated our proposed architecture on the static agent dataset. Finally, we conducted a pilot experiment to measure the performance of adaptive agents in HATs by pairing them with human players.

In the adaptive agent experiment, adaptive agent uses the CEM similarity metric (Sec. 2.4.2) to identify the policy most similar to the human behavior over a fixed number of recent preceding game frames. The frames were tracked using a sliding window, the size of which was adjusted during the hyperparameter tuning phase of experimentation. To perform the adaptation procedure, in each frame, after identifying the most similar agent to the human teammate, the agent referenced the self-play table to select the policy that would best complement the teammate's estimated policy.

In this round of experiment, the five variants including three values of the window size hyperparameter (T in Eq. 2.2) for the adaptive agent (150, 400, 800 frames) and two best-performed static agent policy (representing the extreme condition of 0 window size where the adaptive agent becomes static). Besides that, all experimental settings are the same as in static agent experiment. We got in total, 22 valid data points from human shooters and 25 valid data from human baits.

Fig. 2.11 shows the average team performance of HATs when human players were paired with either static or adaptive agents. We could see from the figure that adaptive agents (marked in orange) have slightly better performance than static agents (marked in yellow), although not statistically significant. In addition, adaptive agents with longer time window (e.g. 800 frames) tend to have better performance in HATs since they accumulate more evidence for human policy inference. However, a larger sample size and and better hyper-parameter tuning might be necessary for future research to confirm the advantage of adaptive agents in HATs.

2.6 Conclusion and Future Work

In this paper, we proposed a novel adaptive agent framework in human agent teaming (HAT) based on the cross-entropy similarity measure and a pre-trained static policy library. The framework was inspired by human teamwork research, which illustrates

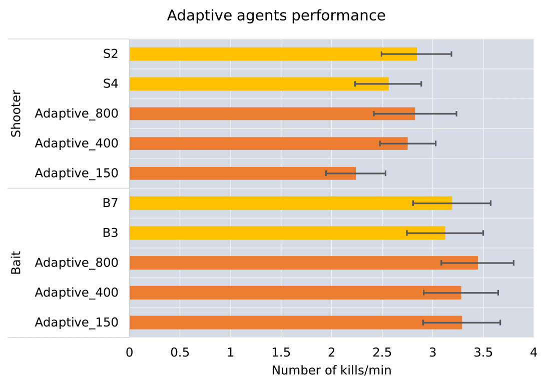

Figure 2.11: Human-agent team performance when humans paired with adaptive or static agent policies. Error bars represent one standard error away from the mean.

important characteristics of teamwork such as the existence of complementary policies, influence of adaptive actions on team performance, and the dynamic human policies in cooperation [38, 40]. Those findings motivate us to introduce an online adaptive agents into HATs in order to maximize the team performance even when given unknown human teammates. The proposed framework adopts a human-model-free method to reduce the computational cost in real-time deployment and make the pipeline more generalizable to diverse task settings and human constraints.

The specific task scenario studied in this paper, i.e. Team Space Fortress, is a nontrivial cooperative game which requires sequential decision-making and real-time cooperation with heterogeneous teammates. We evaluated the validity of proposed adaptive agent framework by running human-agent experiments. Results show that our adaptive agent architecture is able to identify human policies and predict team performance accurately. We constructed a high-dimensional policy space based on exemplar policies in a pre-trained library and leveraged it as a standard and reliable way to categorize and pair human policies. The distance between human policy and the optimal complementary for his/her teammate is shown to be positively correlated with team performance, which confirms the validity of our proposed framework. In additional, we found that human players showed diverse policies in HAT (1) when paired with different teammates (2) over the course of interaction within the same team. These findings point out that we cannot simply impose strong assumptions on humans, e.g. optimality, consistency, and unimodality, prevalent in human-model-based settings. Thus, we employed an online inference mechanism to identify the human policy shifting during the course of interaction and adapt the agent policy in real time.

As for future directions, we would like to enrich the static agent library by introducing novel policies such as imitation learning agents that learn from human demonstrations. A larger coverage in the policy space of exemplar policies library could lead to a more accurate estimation of human policy and a better selection of complementary policy.

Chapter 3

Learning Sparse Discrete Communication Protocols

Recent work in Multi-Agent communication has shown great results in terms of learning communication using agent-agent self play. However, most of these methods learn continuous communication vectors which are exchanged between the agents at a very high frequency. This kind of communication protocol is infeasible in a human-agent team. Humans tend to communicate by composing a finite vocabulary of words. Furthermore, humans also communicate at a frequency much lower than what the agents use during self-play. In this work, we analyse existing work in the area of Learning Sparse Communication protocols using MARL and identify failure cases and propose some solutions to solve them. We also prototype networks to learn discrete communication protocols and demonstrate that they are able to achieve performance comparable to the continuous communication in a predator-prey scenario.

3.1 Introduction

In Multi-Agent scenarios[61], efficient communication is very crucial for completing tasks successfully. In partially observable scenarios, communication enables agents to share information about their observation to the other agents. Recent works[21, 61] have leveraged multi-agent reinforcement learning to learn communication in cooperative[21], mixed[61] and competitive[61] scenarios. Most of the early work in this

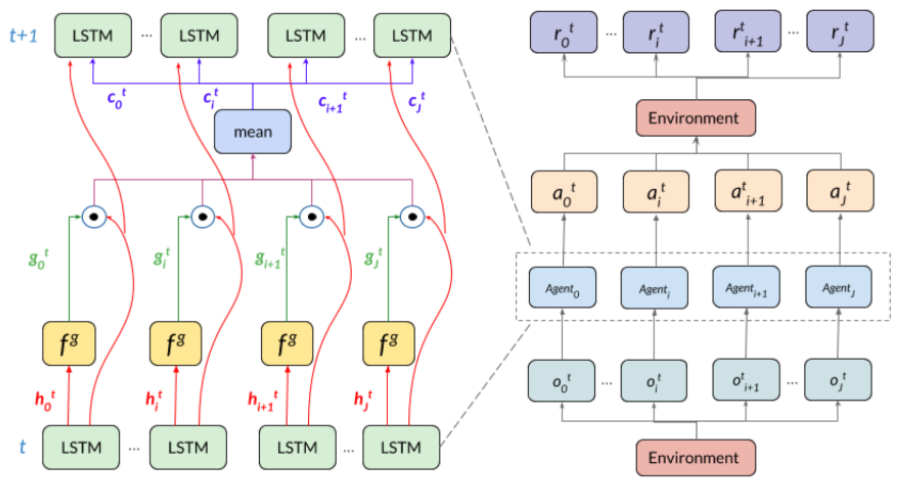

Figure 3.1: IC3 Net Architecture[61]. On the (**Left**) we have an in depth view of how the policy and the gating values are calculated. (**Right**) shows the overall structure of IC3 Net.

area focuses on learning continuous communication vectors which agents communicate with each other at a very high frequency[62] This kind of communication protocol limits the use of this learnt communication in cases an agent might need to communicate with humans. Humans unlike agents can't understand continuous communication vectors. Further, humans can't listen to or speak at such high frequencies which the agents can. Some recent work[61] has focused on learning when to communicate which does help address the issue of high frequency communication. In this work, we analyse the efficiency of the IC3 Net method to learn sparse communication protocols across different kinds of environments. We identify the failure cases of IC3 Net and propose improvements and show how IC3 Net can be adapted to learn sparse and discrete communication protocols.

3.2 Theory

3.2.1 IC3 Net

IC3 Net[61] stands for Individualised Controlled Continuous Communication Model. The overall method is shown in the Fig 3.1. IC3Net[61] uses a recurrent policy and learns a gating function to determine whether to communicate or not. The gating policy and the main policy are both trained using REINFORCE algorithm Lets consider a multi-agent scenario with n agents. We consider an agent j. For the jth agent, its policy is described by

$$h_j^{t+1}, s_j^{t+1} = LSTM(e(o_j^t) + c_j^t, h_j^t, s_j^t)$$

$$a_j^t = \pi(h_j^t)$$

e is the observation encoder network that encodes the observations. Here, c_j^t is the aggregated communication vector that the agent j receives at time t. In addition to the policy each agent also predicts its own gating value which detemines whether or not the agent communicates at that particular time step. $g_j^{t+1} = f^g(h_j^t)$. The communication vector agent j for the next time step is given by,

$$c_j^{t+1} = \frac{1}{J-1} C \sum_{j' \neq j} h_{j'}^{t+1} g_{j'}^{t+1}$$

The same LSTM model is used for all agents, therefore agents share parameters. Both the policy and the gating function is trained using REINFORCE[69]

In our experiments we use IC3 Net in 2 ways:

- **IC3Net-Sparse**: In this case, the gating function g is learnable. This uses continuous communication vectors but could be sparse in time.
- **IC3Net-Fixed**: In this case, the gating function is not learnable and $g = 1$ is fixed. This uses continuous communication vectors and is not sparse in time.

Limitations of IC3 Net

The method formulation and the evaluation metrics used in IC3Net has a number of limitations:

- The sharing of the policy weights makes the method agnostic to the permutation of the agents. However, this might hinder learning in cases where the agents take very different roles within the environment.

- Authors[61] report good results for competitive and mixed scenarios in the paper i.e they were able to show that their method is able to learn policies that achieved high rewards and were able to learn communication protocols that were sparse. However, they don't report such results for fully cooperative scenarios.
- They report sparsity in the form of average sparsity during the episode. However, for practical applications in human-agent teaming it is also crucial that the there is significant time gap between 2 non zero communication vectors. So, its important for us to analyse IC3 Net with respect to this metric as well.
- Even though IC3 Net was shown to perform well in several competitive and mixed scenarios with regards to learning sparse communication protocols, the method doesn't really provide any theoretical guarantees. In some cases, agents might need to work with a constrained communication budget. In such cases, it might be desirable to have a method which is capable of learning policies and communication protocols which can strictly adhere to the communication budgeting constraints.

In this work, we particularly focus on the second point. First we, analyse the performance of IC3 Net in fully cooperative settings. Then we identify failure cases and propose some ways of improvement using an additional gating cost to learn sparse communication protocols especially in cooperative settings.

IC3Net with Gating Cost - IC3NetG

Here we make a small modification in the original IC3Net formulation to try to make it work in cooperative scenarios as well. In cases of cooperative environments, we modify the reward function to penalise non zero communication. We feel this is necessary as in case of cooperative setting, the environment reward doesn't itself provide any incentive for the agents to not communicate all the time. Therefore, we train the agent policy on the following reward function.

$$r_j^t = r_{env}^t + \lambda g_j^t$$

Here λ is a hyperparameter. IC3G uses continuous communication vectors but could be sparse in time.

3.2.2 Discrete Prototype Based Communication

Tucker et. al[64] propose a method of learning learnable discrete tokens for communication using reinforcement learning. In this method, the agent's communication policy is a neural network with the penultimate layer using softmax function with the Gumbel softmax trick[28] which is then multiplied by a learnable matrix T to generate the final communication vector. More formally, the communication policy of the agent is parameterised by neural net weights θ and the learnable matrix T and the communication vector t generated given the agent observation o is given by,

$$\pi = f_\theta(o)$$

$$d = onehot(argmax_i(g_i + \log(\pi_i))$$

$$t = d \times T_{z \times c}$$

In the above equations, z is the number of discrete tokens and c is the dimension of the discrete token.

Tucker et. al[64] in their work demonstrate that prototype based learnt discrete token show higher robustness to noisy communication channels, better human interpretability and better zero shot generalisation. Due to these features, in this work we use this method of learning discrete communication protocols. More specifically we combine this method with with IC3 Net so as to learn communication protocols which are sparse as well as discrete.

The modified IC3Net framework when using discrete prototypes for communication, formally is given by,

$$h_j^{t+1}, s_j^{t+1} = LSTM(e(o_j^t) + c_j^t, h_j^t, s_j^t)$$

$$a_j^t = \pi(h_j^t)$$

$$d_j^t = Proto(h_j^t)$$

$$c_j^{t+1} = \frac{1}{J-1} C \sum_{j' \neq j} d_{j'}^{t+1} g_{j'}^{t+1}$$

3.2.3 IC3Net-Fixed-Proto

In this method, we use IC3 Net-Fixed with discrete prototype based communication. Therefore, in this case the communication would use discrete vectors for communication, however, since, $g = 1$ it won't be sparse in time.

3.2.4 IC3Net-Proto

In this method, we use IC3 Net with discrete prototype based communication. Therefore, in this case the communication would use discrete vectors for communication and the could be sparse in time.

3.2.5 IC3NetG-Proto

In this method, we use both the gating cost as well as the discrete prototype based communication with the aim of learning sparse-discrete communication protocols. Therefore, in this case the communication would use discrete vectors for communication and the could be sparse in time.

3.2.6 Environments

Predator-Prey

We use the environment using by Singh et al[61] as shown in Fig3.2. This task consists of n predators with limited vision trying to find a stationery prey. There are three variants of this environment (1). Cooperative (2). Competitive (3). Mixed. Once the agent reaches the prey it keeps getting a positive reward until the end of the episode. The agents can different degree of vision which is a hyperparameter. There is no loss or benefit from communicating in mixed scenario. In competitive setting, agents get lower rewards if other agents reach the prey and in cooperative setting, reward increases as more agents reach the prey. In all the settings, there is per step negative reward $r = -0.05$ until the agent reaches the prey. In mixed setting, once the agent

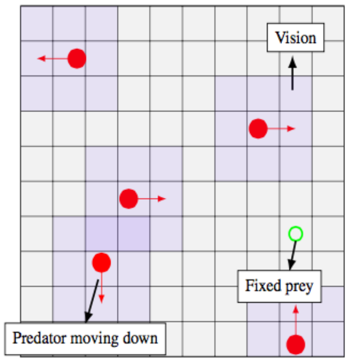

Figure 3.2: Predator-Prey Environment

reaches the prey, it gets a constant positive reward for each step $r = 0.05$ which is independent of whether the other agents have reached the prey or not. In cooperative setting, an agent will get positive reward once reaching the prey, and that reward is given by, $r_{coop} = r_{prey} \times n = 0.05 \times n$, where n is the number of agents on the prey. So, inherently, in cooperative setting, the agent has an incentive to help out other agents. In competitive setting, the agent after reaching the prey gets a reward of $r_{comp} = \frac{0.05}{n}$. For our experiments, we use maximum number of timesteps $t_{max} = 20$, and a 5×5 grid with 3 predators. Each predator can take one of the five actions at each time step: $up, down, left, right, stay$. Predator, prey and all locations on grid are considered unique classes in vocabulary and are represented as one-hot binary vectors. Observation obs, at each point will be the sum of all one-hot binary vectors of location, predators and prey present at that point. With vision of 1, observation of each agent have dimension $3^2|obs|$.

3.3 Experiments

Our experiments are aimed at answering the following questions:

- Is the IC3G-Sparse sufficient for learning sparse communication protocols that also give good performance(high success rate for completing the task) in cooperative predator-prey scenario?
- Does IC3NetG do better than IC3Net-Sparse in case of cooperative scenarios?

- Does IC3Net-Proto-Fixed do as well(in terms of success rate) as IC3Net-Fixed?
- Is IC3NetG-Proto capable of learning sparse-discrete communication protocol that can achieve high success rate?

We measure the success rate and communication rate in our experiments.

- **Success Rate**: For the case of predator prey, success is determined by whether or not all the predators are able to reach the prey within the episode. So success rate, determines the fraction of episodes which are successful.
- **Communication Rate**: For an episode with T timesteps, the communication rate c, can be defined as,

$$c = \frac{g_1 + g_2 + g_3 + .. + g_T}{T}$$

where $g_i \in [0, 1]$ and g_i is the value of the gating function at timestep i.

3.4 Results

In Fig3.3 we compare the success rates achieved by IC3Net-Fixed and IC3Net-Sparse. With IC3Net-Fixed($g = 1$), the agents learn to achieve close to a 100% success rate at convergence. Whereas in the case of a IC3Net-Sparse we see in Fig3.3 how the performance varies across random seeds. Thats also reflected in the communication rate as depicted in 3.4. We observe that for seeds where learnable gating function $g \rightarrow 1$, we achieve success rates with IC3Net-Sparse comparable to IC3Net-Fixed $g = 1$ case. The hidden dimension for the recurrent network is set to 64 in both the cases. We also try the IC3Net-Fixed-Proto method to learn discrete communication vectors for communication instead of continuous ones. We try out $n_{protos} = 25$ and $n_{protos} = 6$. From Fig3.5 we observe that when using 25 prototypes we achieve success rates close to 1 while for $n_{protos} = 6$ the success rates are slightly lower.

For the IC3NetG approach, we find out that the training is extremely sensitive to the value of λ and it is very hard to tune. Here, we report results for λ, $\lambda = 0.01$, where we were able to learn sparse communication protocols that also achieved good success rates as show in Fig3.6. Further, we are able to learn a sparse communication protocol across different seeds. In Fig3.7 we report our results with IC3NetG-Proto

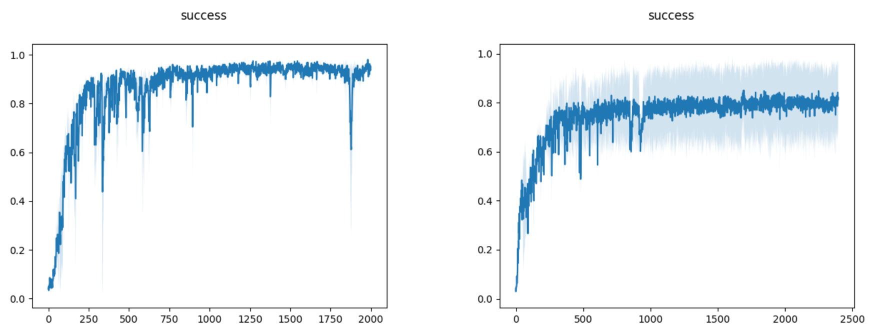

Figure 3.3: (**Left**)Success Rate with IC3Net-Fixed for cooperative predator prey
(**Right**) Success Rate with IC3Net-Sparse in cooperative predator-prey

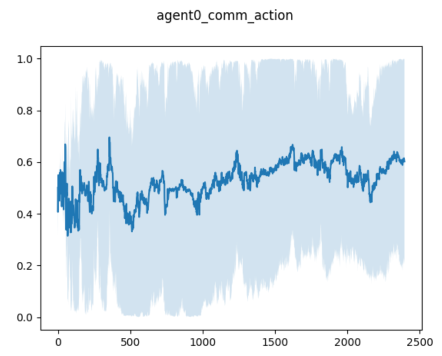

Figure 3.4: Communication rate with with IC3Net-Sparse in cooperative predator-prey

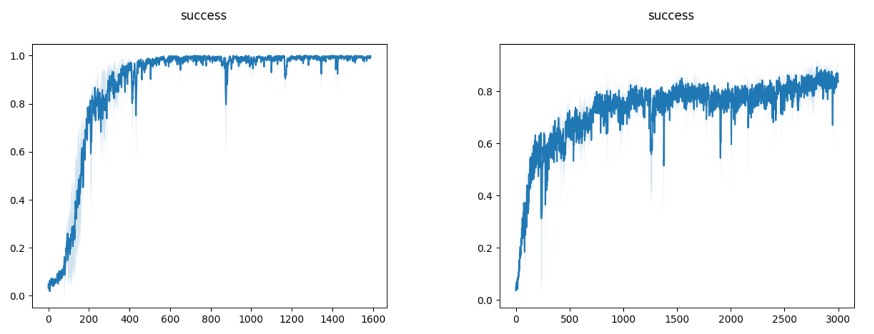

Figure 3.5: (**Left**)Success Rate with IC3Net-Fixed-Proto using 25 prototypes in cooperative predator-prey
(**Right**) Success Rate with IC3Net-Fixed-Proto using 8 prototypes in cooperative predator-prey

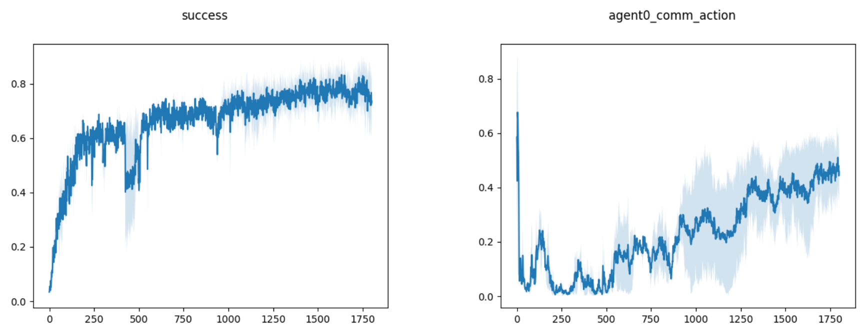

Figure 3.6: (**Left**)Success Rate using IC3NetG with $\lambda = 0.01$, (**Right**) Communication Rate IC3NetGwith $\lambda = 0.01$

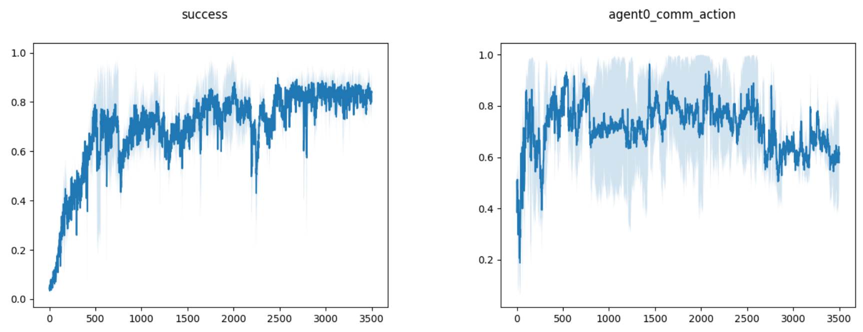

Figure 3.7: (**Left**)Success Rate using IC3NetG-Proto, (**Right**) Communication Rate IC3NetG-Proto

where we use 25 learnable discrete prototypes for communicate along with a gating penalty to encourage sparse communication. In this case too, we achieve good success rates and a sparse communication protocol.

3.5 Conclusions and Future Work

In this work, we attempt at solving 2 crucial problems for learning interpretable communication: (1). Learning to communicate using discrete tokens (2). Learning to communicate in a sparse manner. For discrete communication, we show how IC3Net-Proto is able to achieve success rates comparable to IC3Net-Fixed showing that we could work with a limited number of tokens and achieve similar performance as when using continuous vectors for communication. For sparse communication, we showed how IC3Net in its original formulation is not sufficient to learn sparse communication protocols in fully cooperative scenarios. Therefore, we proposed IC3NetG which uses a gating penalty to limit communication. Although, we did demonstrate desirable outcomes with a few values of λ, we experienced that it was very difficult to tune the value of λ. For future work, we would like to come up with methods which could automatically tune the value of λ. Further, we would also like to take a more principled look at the problem of sparse communication by setting a hard constraint on the communication budget.

Chapter 4

Conclusions and Future Work

Conclusions

This thesis presented methods for robust Imitation Learning, Adaptation in Human-Agent Teaming and Learning Sparse-Discrete Communication protocols using Multi-Agent Reinforcement Learning. The ability to be able to imitate humans is very essential for designing human like agents. Therefore, addressing the shortcomings of Generative Adversarial Imitation Learning, a state of the art method for Imitation Learning is very important. In this work, we show how our method of neutral rewards overcame the reward bias problem of Generative Adversarial Imitation Learning. Our work on Adaptation in Human-Agent Teaming demonstrates a method of designing adaptative agents which can utilise an agent library to adapt to new human policies during test time. We believe that this is of great significance for a variety of practical applications. Our method is not restrictive in terms of what the policies in the agent library are. So, an adaptative AI agents could leverage diverse imitation learning policies too, to measure the similarity of new human policy during test to the imitation learning policies which were learnt using past interactions. The similarity metric could then be utilised in order to find the most optimal partner to the human policy from the agent library. Our work on Learning Sparse-Discrete Communication protocols, tries to achieve two very desirable properties for any interpretable communication protocol: Sparsity and Discreteness. Overall, through this work we try to address three very fundamental problems for designing smart AI agents: (1). The ability to imitate humans (2) The ability to adapt to perceived changes in human behavior (3). The ability to communicate in a manner which is interpretable.

Future Work

It's crucial to evaluate our method of using neutral rewards in Generative Adversarial Imitation Learning on real tasks like manipulation and social navigation in future. Further, one major issue in Generative Adversarial Imitation Learning under the different formulations of the reward is that the agent doesn't have the idea of the goal state of the expert demonstration. So, we would also like to compare our proposed method of using neutral rewards against goal conditioned GAIL methods. In our work on adaptation we rely on an agent library for calculating the similarity metric which are then used for real time adaptation. This agent library in our case consisted of rule based as well RL trained agents which in turn use knowledge about the game. To scale our approach to new scenarios with a larger number of agents we would like to create this policy library purely from demonstrations and not using explicit domain knowledge about the environment. Methods like INFO-GAIL[42] could help us learn the imitation learning policy and also help capture the diversity in demonstrations as the imitation learnt policy is conditioned both on state and a latent variable which could potentially capture the diversity of human policies without requiring environment specific information. In our work on learning sparse communication, a more rigorous theoretical analysis is required particularly in cases where we might have a fixed budget for communication. For this we could potentially use methods from safe RL like CPO[2] and model the communication budget constraints as being analogous to safety constraints in safe-RL. Further, we didn't consider environments with heterogeneous agents in our current work. It might be interesting to evaluate whether in competitive scenarios with heterogeneous agents can we still just rely on the environment reward to induce sparsity in the communication protocol. We would also like to compare our method of learning sparse communication in cooperative settings to some recent work by Goyal et Al[22] who learn sparse communication protocols by using variational information bottleneck in the communication channel.

Bibliography

[1] Pieter Abbeel and Andrew Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, 09 2004. 1.2

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization, 2017. 4

[3] Akshat Agarwal, Ryan Hope, and Katia Sycara. Challenges of context and time in reinforcement learning: Introducing space fortress as a benchmark. *arXiv preprint arXiv:1809.02206*, 2018. 2.1

[4] Siddharth Agrawal and Mary-Anne Williams. Robot authority and human obedience: A study of human behaviour using a robot security guard. In *Proceedings of the companion of the 2017 ACM/IEEE international conference on human-robot interaction*, pages 57–58, 2017. 2.1

[5] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018. 2.2

[6] Michael Bain. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995. 2.1, 2.4.2

[7] Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning from physical human corrections, one feature at a time. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 141–149, 2018. 2.2

[8] Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *AAAI*, volume 15, pages 2010–2016. Citeseer, 2015. 2.2

[9] Jeffrey M Bradshaw, Paul Feltovich, and Matthew Johnson. Human-agent interaction. 2011. 2.1

[10] Jessie YC Chen and Michael J Barnes. Supervisory control of multiple robots: Effects of imperfect automation and individual differences. *Human Factors*,

54(2):157–174, 2012. 2.1

[11] Jessie YC Chen and Michael J Barnes. Human–agent teaming for multirobot control: A review of human factors issues. *IEEE Transactions on Human-Machine Systems*, 44(1):13–29, 2014. 2.1

[12] Jessie YC Chen, Shan G Lakhmani, Kimberly Stowers, Anthony R Selkowitz, Julia L Wright, and Michael Barnes. Situation awareness-based agent transparency and human-autonomy teaming effectiveness. *Theoretical issues in ergonomics science*, 19(3):259–282, 2018. 2.2

[13] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. 1.4

[14] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018. 1.1, 1.3.6

[15] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009. 1.2

[16] Mustafa Demir, Nathan J McNeese, and Nancy J Cooke. Team synchrony in human-autonomy teaming. In *International Conference on Applied Human Factors and Ergonomics*, pages 303–312. Springer, 2017. 2.1

[17] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Advances in Neural Information Processing Systems*, pages 15324–15335, 2019. 1.2, 1.3.3

[18] Xiaocong Fan, Michael McNeese, Bingjun Sun, Timothy Hanratty, Laurel Allender, and John Yen. Human–agent collaboration for time-stressed multicontext decision making. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 40(2):306–320, 2009. 2.2

[19] Jaime F Fisac, Eli Bronstein, Elis Stefansson, Dorsa Sadigh, S Shankar Sastry, and Anca D Dragan. Hierarchical game-theoretic planning for autonomous vehicles. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9590–9596. IEEE, 2019. 2.1, 2.2

[20] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017. 1.1, 1.2

[21] Anirudh Goyal, Yoshua Bengio, Matthew Botvinick, and Sergey Levine. The variational bandwidth bottleneck: Stochastic evaluation on an information budget, 2020. 3.1

[22] Anirudh Goyal, Yoshua Bengio, Matthew Botvinick, and Sergey Levine. The variational bandwidth bottleneck: Stochastic evaluation on an information budget, 2020. 4

[23] C Shawn Green and Daphne Bavelier. Enumeration versus multiple object tracking: The case of action video game players. *Cognition*, 101(1):217–245, 2006. 2.1

[24] William H Guss, Cayden Codel, Katja Hofmann, Brandon Houghton, Noboru Kuno, Stephanie Milani, Sharada Mohanty, Diego Perez Liebana, Ruslan Salakhutdinov, Nicholay Topin, et al. The minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019. 2.1

[25] Maaike Harbers, Catholijn Jonker, and Birna Van Riemsdijk. Enhancing team performance through effective communication. 2012. 2.2

[26] T Haynes and Sandip Sen. Co-adaptation in a team. *International Journal of Computational Intelligence and Organizations*, 1(4):1–20, 1996. 2.1

[27] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc., 2016. 1.1, 1.2, 1.3.1, 1.3.2, 1.3.3, 1.4

[28] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017. 3.2.2

[29] Wonseok Jeon, Seokin Seo, and Kee-Eung Kim. A bayesian approach to generative adversarial imitation learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7429–7439. Curran Associates, Inc., 2018. 1.2

[30] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000. 2.4.1

[31] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *International Conference on Learning Representations*, 2019. 1.1, 1.2, 1.3.2, 1.3.4, 1.3.6, 1.4

[32] Steve WJ Kozlowski and Georgia T Chao. Unpacking team process dynamics and emergent phenomena: Challenges, conceptual advances, and innovative methods. *American Psychologist*, 73(4):576, 2018. 2.1, 2.2

[33] Steve WJ Kozlowski, James A Grand, Samantha K Baard, and Marina Pearce. Teams, teamwork, and team effectiveness: Implications for human systems

integration. 2015. 2.1, 2.2

[34] Steve WJ Kozlowski and Katherine J Klein. A multilevel approach to theory and research in organizations: Contextual, temporal, and emergent processes. 2000. 2.1, 2.2

[35] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE, 2017. 1.1

[36] Vitaly Kurin, Sebastian Nowozin, Katja Hofmann, Lucas Beyer, and Bastian Leibe. The atari grand challenge dataset. *arXiv preprint arXiv:1705.10998*, 2017. 2.1

[37] Steven James Levine and Brian Charles Williams. Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. *Journal of Artificial Intelligence Research*, 63:281–359, 2018. 2.2

[38] Huao Li, Dana Hughes, Michael Lewis, and Katia Sycara. Individual adaptation in teamwork. In *Proceedings of the 42nd Annual Conference of the Cognitive Science Society*, page In Press, 2020. 2.6

[39] Huao Li, Stephanie Milani, Vigneshram Krishnamoorthy, Michael Lewis, and Katia Sycara. Perceptions of domestic robots' normative behavior across cultures. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 345–351, 2019. 2.1

[40] Huao Li, Tianwei Ni, Siddharth Agrawal, Dana Hughes, Michael Lewis, and Katia Sycara. Team synchronization and individual contributions in coop-space fortress. In *Proceedings of the 64th Human Factors and Ergonomics Society Annual Meeting*, page In Press, 2020. 2.1, 2.4.1, 2.6

[41] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3812–3822, 2017. 1.2

[42] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations, 2017. 4

[43] Amir Mané and Emanuel Donchin. The space fortress game. *Acta psychologica*, 71(1-3):17–22, 1989. 2.3

[44] Nathan J McNeese, Mustafa Demir, Nancy J Cooke, and Christopher Myers. Teaming with a synthetic teammate: Insights into human-autonomy teaming. *Human factors*, 60(2):262–273, 2018. 2.1

[45] Ben B Morgan Jr et al. Measurement of team behaviors in a navy environment. final report. 1986. 2.1

[46] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Prze-

myslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 2.1

[47] Raja Parasuraman and Victor Riley. Humans and automation: Use, misuse, disuse, abuse. *Human factors*, 39(2):230–253, 1997. 2.1

[48] Sid Reddy, Anca Dragan, and Sergey Levine. Where do you think you're going?: Inferring beliefs about dynamics from behavior. In *Advances in Neural Information Processing Systems*, pages 1454–1465, 2018. 2.2

[49] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010. 1.2

[50] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. 1.2

[51] Dorsa Sadigh, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017. 2.2

[52] Dorsa Sadigh, Nick Landolfi, Shankar S Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state. *Autonomous Robots*, 42(7):1405–1426, 2018. 2.1, 2.2

[53] Dorsa Sadigh, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan. Information gathering actions over human internal state. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 66–73. IEEE, 2016. 2.1

[54] Eduardo Salas, Nancy J Cooke, and Michael A Rosen. On teams, teamwork, and team performance: Discoveries and developments. *Human factors*, 50(3):540–547, 2008. 2.1, 2.5.2

[55] Eduardo Salas, Dana E Sims, and C Shawn Burke. Is there a "big five" in teamwork? *Small group research*, 36(5):555–599, 2005. 2.1

[56] Jean Scholtz. Theory and evaluation of human robot interactions. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*,

pages 10–pp. IEEE, 2003. 2.1

[57] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015. 2.4.1

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. 1.4

[59] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2.4.1

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. 2.4.1

[61] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks, 2018. (document), 3.1, 3.1, 3.2.1, 3.2.1, 3.2.6

[62] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016. 3.1

[63] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018. 1.2

[64] Mycal Tucker, Huao Li, Siddharth Agrawal, Dana Hughues, Katia P. Sycara, Michael Lewis, and Julie Shah. Emergent discrete communication in semantic spaces. *NeurIPS 2021, Under Review.* 3.2.2

[65] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015. 2.4.1

[66] Emma M van Zoelen, Anita Cremers, Frank PM Dignum, Jurriaan van Diggelen, and Marieke M Peeters. Learning to communicate proactively in human-agent teaming. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 238–249. Springer, 2020. 2.2

[67] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 2.1

[68] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. 2.1

[69] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992.

3.2.1

[70] Mehrdad Zakershahrak, Akshay Sonawane, Ze Gong, and Yu Zhang. Interactive plan explicability in human-robot teaming. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1012–1017. IEEE, 2018. 2.1