Learning to play collaborative-competitive games

Kshama Dwarakanath
S. Shankar Sastry

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-201
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-201.html

December 16, 2020

Acknowledgement

I am extremely grateful to having had the opportunity to work closely with Prof. Shankar Sastry, who has been one of my role models in control theory ever since my undergraduate days. I am especially thankful to Prof. Sastry for encouraging me to try out different projects to find an area of interest. He has been very resourceful and open with regard to all my decisions in graduate school. I also want to thank Prof. Yi Ma for motivating me to try out state of the art reinforcement learning algorithms for non-holonomic systems, thereby introducing me to the field of applied reinforcement learning.

Learning to play collaborative-competitive games

by

Kshama Dwarakanath

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Fall 2020

Abstract

Learning to play collaborative-competitive games

by

Kshama Dwarakanath

Master of Science in Computer Science

University of California, Berkeley

Professor Shankar S. Sastry, Chair

In this project, we formalize a collaborative-competitive game with competition between two teams and collaboration between members of each team. The players from the first team seek to reach their individual goals while avoiding capture by the second team. And, the second team seeks to capture all players in the first team. The competition between the two teams arises from the fact that the second team seeks to capture the first team's players, while the latter seek to reach their individual goals while avoiding capture. The players within each team can collaborate with each other in order to achieve their individual and team goals. The ground rules for game play are cast in the form of a Markov Decision Process with the goal of learning optimal game play strategies for members of the first team. We collect expert trajectories from human experts that played the game, and use this data to learn similar game play strategies designed to ensure that the first team wins the game. A recent approach for imitation learning called Generative Adversarial Imitation Learning (GAIL) is examined in the context of these collaborative-competitive games. The results of running GAIL on expert data are contrasted against those got from state of the art algorithms from the domain of imitation learning as well as (forward) reinforcement learning. We see that the learnt policies resemble in logic to those used by human experts in playing the game, while being successful in about 70% of new games played. This success rate is very close to that of the human experts playing the game.

Contents

List of Figures

List of Tables

Acknowledgments

I am extremely grateful to having had the opportunity to work closely with Prof. Shankar Sastry, who has been one of my role models in control theory ever since my undergraduate days. I am especially thankful to Prof. Sastry for encouraging me to try out different projects to find an area of interest. He has been very resourceful and open with regard to all my decisions in graduate school. I also want to thank Prof. Yi Ma for motivating me to try out state of the art reinforcement learning algorithms for non-holonomic systems, thereby introducing me to the field of applied reinforcement learning.

Chapter 1

Introduction and Related Work

Multi-agent systems have exceedingly become popular with the advent of robot teams, autonomous driving, drone delivery [6] and various other applications involving decentralized control. A multi-agent system is comprised of agents that interact with each other and with a common environment on which they can act through their actuators [22]. The complexity of many tasks arising in multi-agent systems results in difficulty in a priori design of effective agent behaviour [5]. The interaction between agents agents in a multi-agent system can be one of three types - competitive, cooperative, or mixed. In competitive settings, we have agents that each seek to achieve their goals independent of other agents. In cooperative settings, all agents work together towards a common goal. In the mixed setting, the agents each have their own goal along with a system goal that they need to work towards. In this report, we are interested in a special case in the third category where there exist two competing teams of agents with their respective team goals. The agents in each team need to cooperate with other members of their own team to achieve their team goal while they compete with members from the other team.

The complex nature of agent behaviours in multi-agent systems makes analytical control design for such systems complex. Such systems often benefit from the use of trial-and-error based approaches, such as reinforcement learning, to learning optimal agent behaviours[18]. The method of reinforcement learning (RL) involves modeling the agent in the environment as a Markov Decision Process (MDP) with an underlying transition dynamics and (if available) a reward function or cost function to encode the task of interest. In such a setup, the agent learns by reinforcement of its action through the reward function. Most of the successes of RL have been in single agent domains where the goal is to find the best decision making behaviour for a single agent in an environment with no other agents [11]. On the contrary, the technique of reinforcement learning (RL) is especially suited to these multi-agent systems in which agents can learn to behave optimally from their interaction with other agents and the environment.

While RL is certainly a viable approach to learning to play collaborative-competitive games, a fundamental assumption that underlies its use is the availability of a reward function or cost function that determines how good a certain agent behaviour is. On the other hand, humans often learn to perform tasks via imitation - they observe others perform a task, and then very quickly infer the appropriate actions to take based on their observations [20]. This is of interest in applications such as multi-robot control and training groups of robots/humans, where a notion of desired behaviour is better known than the exact reinforcement signal that could give rise to that behaviour. Imitation learning refers to this idea of observing expert behaviour in terms of their policies or trajectories with the goal of learning to perform the task. One approach to solving imitation learning problems is called behavior cloning where one simply formulates this problem as a supervised learning problem where we learn a mapping from states to action from the given expert demonstrations. And, the learner replays this learnt policy in new test cases/states as well. This relies on covering all state-action pairs in the training data so as to be able to predict test behaviour well.

Another approach to solving imitation learning problems is by first inverse learning rewards from expert trajectories and then extracting a policy from that reward function with reinforcement learning. Inverse Reinforcement Learning (IRL) refers to the problem of extracting the reinforcement signal or the reward function given observed, optimal behaviour. There have been numerous approaches proposed to solve the IRL problem. One of the first and most cited papers in this field is [14] wherein lies the foundation of IRL as studied in the field of machine learning. The authors formulate the IRL problem as one of finding rewards that give rise to expert policies as solutions to reinforcement learning problems with these rewards.

Standard IRL is ill-posed since the zero reward function is always a solution, which in turn means that all policies are optimal! Hence, all attempts at IRL involves enforcing further constrains or modifying the IRL objective so as to prevent this degeneracy. [1] involves matching expected feature counts between the demonstrated examples and the learner's policy. The underlying assumption made here is that the expert agent is acting near-optimally in an MDP, while the learner is capable of nearly matching feature expectations. In [15], the authors derive a framework that looks at learning a reward function that makes the expert policy *much better* (in terms of rewards) than any other policy by a margin that scales with the difference between the two policies. They also make the comment that the distinction between [1] and [15] is evocative of generative versus discriminative learning where the former makes stronger assumptions to derive the process that generates expert behaviour. And the latter tries to mimic expert behaviour while being agnostic about the underlying process that generates it.

In [25], the authors adopt a probabilistic approach to reasoning about uncertainty in inverse reinforcement learning. This uncertainty is deemed to arise from noise and imperfect expert behaviour that is commonplace when the expert agents are humans. They utilize the principle of maximum entropy to resolve ambiguity in choosing a decision distribution under the constraint of matching the reward value of demonstrated behavior. A common disadvantage of all of the above approaches to imitation is the need to perform IRL to recover the expert's cost function and then using RL to extract a policy from that cost function, which can be slow. A recent approach to imitation learning called generative adversarial imitation learning (GAIL) aims at overcoming this problem by directly extracting a policy given expert data[9]. And, we will look at this approach in more detail with specific focus on learning to play collaborative-competitive games.

In this report, we'll focus on a specific class of collaborative-competitive games with two competing teams of players, called Team Human and Team Computer respectively. The human team players seek to reach their individual goals while avoiding capture by the computer team. And, the computer team seeks to capture all players in the human team. The competition between the two teams arises from the fact that the computer team seeks to capture the human players, while the human players seek to reach their individual goals while avoiding capture. The players within each team can collaborate with each other in order to achieve their individual and team goals.

The goal of this project is to learn optimal game play strategies for members of a team under a fixed (but unknown) strategy adopted by the competing team. In order to do this, we first devise the rules of the game in terms of allowed team behaviour and formulate the game as a Markov Decision Problem with appropriate state and action spaces in Chapter 3 according to the definitions given in Chapter 2. The MDP formulation makes the problem suited for the use of state of the art imitation learning algorithms. In particular, we look at a recent imitation learning algorithm called Generative Adversarial Imitation Learning (GAIL) that finds a policy *close* to that of the expert, without going through the cycle of IRL followed by RL for imitation learning[9]. Chapter 3 also talks about the composition of the GAIL algorithm. We then contrast the performance of GAIL for our game to that of behavioural cloning as well as state of the art reinforcement learning algorithms such as Trust Region Policy Optimization, Deep Q Networks and Actor Critic methods in Chapter 4. This is followed by a discussion of the results in Chapter 5.

Chapter 2

Preliminaries

Definition 1. *For any positive integer n, denote the set $\{1, 2, \cdots, n\}$ by $[n]$.*

Definition 2 (Convex conjugate of a function)**.** *Given a function $f : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R} \cup \{\infty\}$, its convex conjugate $f^\star : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R} \cup \{\infty\}$ is defined as*

$$f^\star(x) := \sup_{y \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}} x^T y - f(y)$$

Definition 3 (MDP)**.** *A finite Markov Decision Process (MDP) is a tuple $\big(\mathcal{S}, \mathcal{A}, \{P_a : a \in \mathcal{A}\}, c, \gamma\big)$ where*

- *$\mathcal{S} = \{1, 2, \cdots, n\}$ is a finite set of states*
- *$\mathcal{A} = \{1, 2, \cdots, m\}$ is a finite set of actions*
- *For each action $a \in \mathcal{A}$, $P_a \in \mathbb{R}^{n\times n}$ is a state transition matrix where $P_a(i, j)$ denotes the probability of transitioning into state j upon applying action a in state i*
- *$c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the cost function where $c(i, j)$ is the cost accrued from taking action j when in state i* [1]
- *$\gamma \in [0, 1)$ is the discount factor to weigh down future rewards with respect to current rewards*

Note that for any MDP with finite state and action spaces, one can always number the states and actions as in definition 3.

Definition 4 (Policy)**.** *A policy is a map $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$ from the current state to the set of all probability distributions on $\mathcal{A}$ denoted by $\mathbb{P}(\mathcal{A})$. Since $\mathcal{A}$ is a finite set, $\mathbb{P}(\mathcal{A})$ is essentially the set of all vectors in $[0, 1]^m$ with components summing to 1.*

[1] Note that we look at MDPs with a cost function so that the goal is to minimize the expected sum of discounted costs. This is equivalent to an MDP with reward function defined as the negative of the cost function, where the goal is to maximize the expected sum of discounted rewards.

Let $\Pi \subseteq \{\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})\}$ be the set of all policies of interest.

Definition 5 (Cost of a policy). *The cost of a policy $\pi \in \Pi$ under cost function $c : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ is denoted by $\mathbb{E}_\pi[c(s,a)]$ and defined as*

$$\mathbb{E}_\pi[c(s,a)] := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t,a_t) \middle| a_t \sim \pi(\cdot|s_t)\right]$$

Definition 6 (Entropy of a policy). *The entropy of a probabilistic policy π is denoted by $H(\pi)$ and defined as*

$$H(\pi) := \mathbb{E}\big[-\log \Psi\big] \tag{2.1}$$

where Ψ is a random variable drawn from the distribution given by $\pi(\mathcal{S})$.

Definition 7 (Occupancy measure of a policy). *Define the occupancy measure $\rho_\pi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ For a policy $\pi \in \Pi$ as*

$$\rho_\pi(s,a) := \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s|\pi) \tag{2.2}$$

With this definition of the occupancy measure, one can express the cost of policy $\pi \in \Pi$ as

$$\mathbb{E}_\pi[c(s,a)] = \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \rho_\pi(s,a) c(s,a)$$

Problem (RL). *Given a cost function $c : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$, the goal in reinforcement learning is to extract from it a policy denoted by $\mathrm{RL}(c)$ so that*

$$\mathrm{RL}(c) \in \arg\min_{\pi\in\Pi} \mathbb{E}_\pi[c(s,a)] \tag{2.3}$$

Problem (IRL). *Given an expert policy $\pi_E \in \Pi$ that we want to rationalize with inverse reinforcement learning, we wish to find a cost function $c : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ such that*

$$\pi_E \in \arg\min_{\pi\in\Pi} \mathbb{E}_\pi[c(s,a)] \tag{2.4}$$

The goal in inverse reinforcement learning (IRL) is to extract a cost function so that an optimal policy in an MDP with this cost function imitates the expert policy. In maximum entropy IRL [25], we want to find a cost function that that assigns low cost to the expert policy and high cost to all other policies while not overfitting on the expert data. This is realized by ensuring high entropy for the expert policy.

Problem (Maximum Entropy IRL)**.** *Given an expert policy $\pi_E \in \Pi$ that we want to rationalize with maximum entropy IRL, we wish to find a cost function $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that*

$$\pi_E \in \arg\min_{\pi \in \Pi} \mathbb{E}_\pi[c(s,a)] - H(\pi) \tag{2.5}$$

(2.5) implies that a solution to the maximum entropy IRL problem satisfies

$$\min_{\pi \in \Pi} \mathbb{E}_\pi[c(s,a)] - H(\pi) - \mathbb{E}_{\pi_E}[c(s,a)] + H(\pi_E) \geq 0 \tag{2.6}$$

Among all cost functions $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that satisfy (2.6), we pick one that maximizes the difference between the expert policy and all other policies:

$$c^\star \in \arg\max_c \min_{\pi \in \Pi} \mathbb{E}_\pi[c(s,a)] - H(\pi) - \mathbb{E}_{\pi_E}[c(s,a)] + H(\pi_E)$$

That is, the solution to our maximum entropy IRL problem is a cost function $c^\star : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that

$$c^\star \in \arg\max_c \left[\min_\pi \mathbb{E}_\pi[c(s,a)] - H(\pi) \right] - \mathbb{E}_{\pi_E}[c(s,a)] \tag{2.7}$$

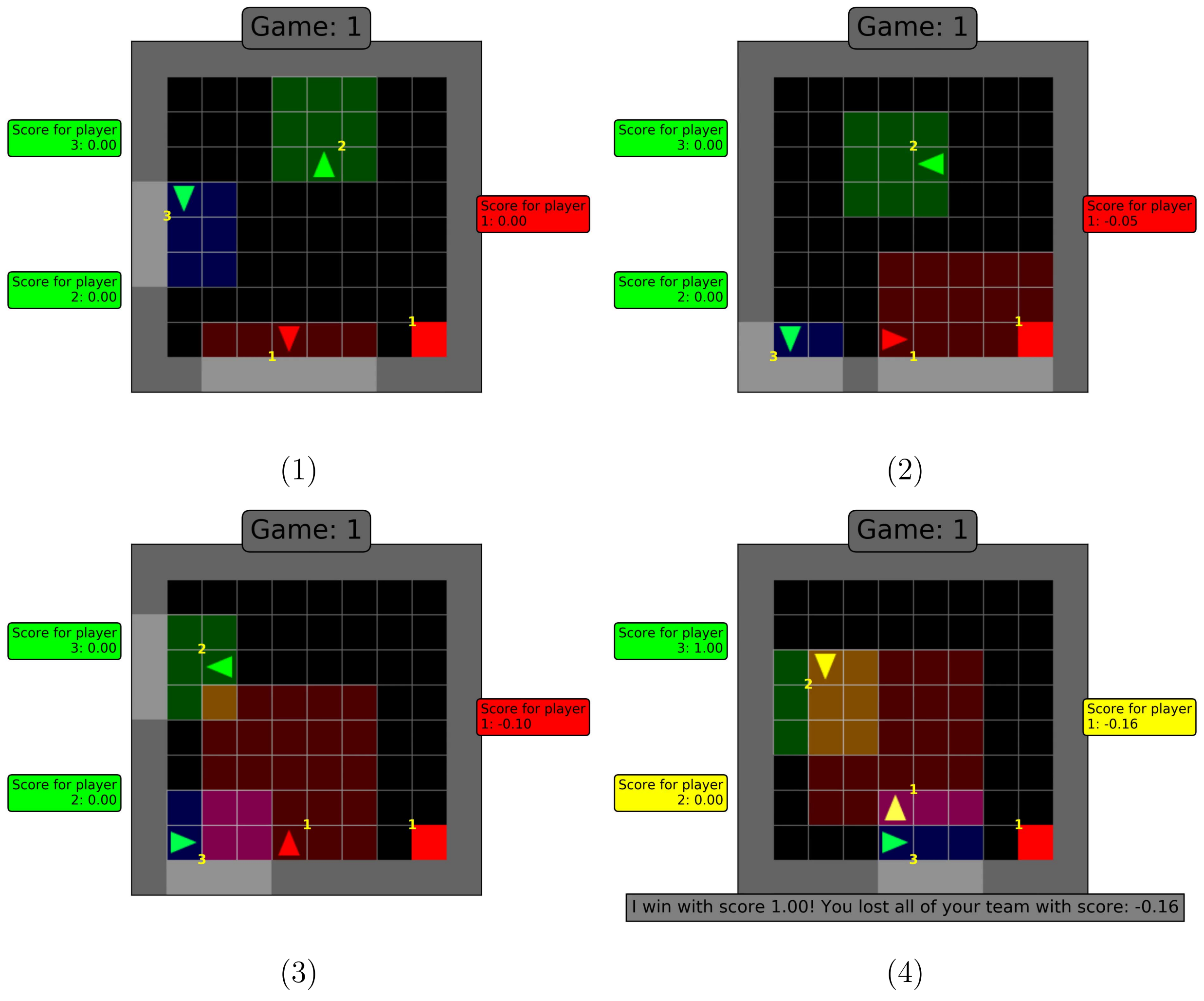
where the square brackets indicate that the internal optimization corresponds to solving an RL problem with cost c, while also maximizing the entropy of the optimal policy.

Chapter 3

Problem Formulation

In this chapter, we describe the class of collaborative-competitive games of interest and set some ground rules for game play. We consider a setup with two competing teams of players, called Team Human and Team Computer with n_H and n_C players respectively. Each player in the human team has an associated goal to be reached. The goal of the human team is to ensure that its players reach their individual goals while avoiding capture by the computer team. The goal of the computer team is to capture all players in the human team. The competition between the two teams arises from the fact that the computer team seeks to capture the human players, while the human players seek to reach their individual goals while avoiding capture. The players within each team can collaborate with each other in order to achieve their individual and team goals.

In order to motivate the need for collaboration within players of each team, consider the following example with $n_H = 1$ and $n_C = 2$. The computer players have a square region of size 3 as their capture zone. Thus, any human player that enters this region is captured by the computer player. Similarly, the human player has a capture region of size 5. This advantage in the size of the capture region for the human team is counteracted by a disadvantage in speed. While the human player can move only 1 grid at a time, the computer players can each move up to 3 and 5 grids at a time respectively. Assume that the strategy of the human player is to get to its goal (the red grid at the bottom right of Figure 3.1) while avoiding capture by computer players, by capturing any computer player that gets close to it. In the first case, assume that the computer players collaborate with each other to capture the human player as in Figure 3.1. What happens here is that the computer team decides to provide its player 2 as a bait for the human player. As the human player starts to chase computer player 2, computer player 3 attacks it from behind and captures it as shown in the sequence of snapshots in Figure 3.1. In the second case, assume that the players of the computer team do not collaborate with each other to capture members of the human team. Then, one possible non-collaborative strategy for the computer players is to both chase the human player by taking the shortest path to it. Such a non-collaborative strategy could result in a loss for the computer team as opposed to that with the collaborative strategy described before.



I win with score 1.00! You lost all of your team with score: -0.16

Figure 3.1: The need for collaboration

From the example above, we see that the presence of collaboration within members of a team greatly improves the team's competitive ability against its opposition. The goal of this project is to learn optimal game play strategies for members of a team under a fixed (but unknown) strategy adopted by the competing team. In order to do this, we first devise the rules of the game in terms of permitted team behaviour. This qualitative description of the game is then translated into its formulation as a Markov Decision Problem with appropriate state and action spaces.[1] The MDP formulation makes the problem suited for the use of

[1]Note that in this section we look at MDPs with a reward function so that the goal is to maximize the expected sum of discounted rewards. This is because of the interpretation of the reward as a score used to generate human game play data. This is equivalent to studying an MDP with cost function being the negative reward function.

state of the art reinforcement learning and imitation learning algorithms. In particular, we focus on a recent imitation learning algorithm called generative adversarial imitation learning (GAIL) that seeks to extract a policy from expert data, as if it were obtained by reinforcement learning following inverse reinforcement learning. The composition of the GAIL algorithm is described later in this chapter.

3.1 Rules of the game

- We have two competing teams: Team Human and Team Computer, with n_H and n_C players respectively.
- Assume the human players are numbered $1, 2, \cdots, n_H$ and the computer players are numbered $n_H + 1, \cdots, n_H + n_C$.
- The goal of the human team is to reach their individual goals.
- The goal of human player $i \in [n_H]$ is denoted by $G_i = (x_{G_i}, y_{G_i})$.
- The goal of the computer team is to capture all players in the human team.
- The game is set in a grid world environment of width w and height h.
- The state of each player is defined by its position in the grid world, and its orientation which can be one of 4 possible orientations - north, east, south and west.
- The state (position, orientation) of all players are visible to all other players.
- Each player has a capture window aligned with its orientation. Any other player that enters this capture zone is captured.
- Each player i can either turn left by 90°, turn right by 90°, stay put or move forward by v_i number of grids where v_i is the velocity of player i.
- Each computer player has velocity v_i that is greater than or equal to that of the human team.
- The human team has capture window of size greater than or equal to that of the computer team.
- Reward/Score structure:[2]
    1. Scores of all players are displayed on the game screen. All players start with 0 score.
    2. R immediate points for reaching goal for human players.

[2]This is equivalent to having a cost function being the negative reward/score function.

3. C points added for capturing competitor (given at goal for human players and immediately for computer players).
4. C immediate points reduced for being captured by competitor.
5. The score of a human player is decremented by the negative l_1 from its current grid position to its goal. This is done to continuously provide reward signals to the human players to direct them towards their goals.
6. The scores of human players are decremented every time step taken to get to goal. This is so that faster maneuvers to goal are preferred. It also helps to prevent the strategy of always capturing all computer players before going to the goal.

- The computer team wins if they capture all human players.
- The human team wins when there is at least one of its players at their goal and all other players are captured.
- The game ends when one of the following occurs:
  1. The human teams wins.
  2. The computer team wins.
  3. Maximum game time is exceeded, in which case the computer team wins.

Exemplar pictures

Figures 3.2-3.4 are exemplar pictures for the game described above. The human player/s are colored red and the computer players are colored green. The yellow number alongside each player denotes the index of that player. The orientation of each player is the direction in which the triangle depicting each player points. The red boxes indicate goal locations for the human team (indexed by the index of human players). The rectangular region extending from the base of each player is the capture region for that player. Once a player is captured by a competitor, it turns yellow. For instance in Figure 3.3, there are 3 computer players and 1 human player, with computer player 2 captured by human player 1 and hence, in yellow. The capture regions of all computer players are squares of size 3, while the human players have capture regions of sizes 7 and 5 respectively in Figure 3.2. The scores for all computer players are displayed along the left edge of the grid world, while those of the human players are displayed along the right edge. Any messages to the human experts playing the game on behalf of the human team are displayed below the grid world as in Figure 3.3.

3.2 MDP formulation

In order to find optimal policies for both teams in a collaborative-competitive game, we adopt an iterative framework that learns policies for one team while holding that of the competitor

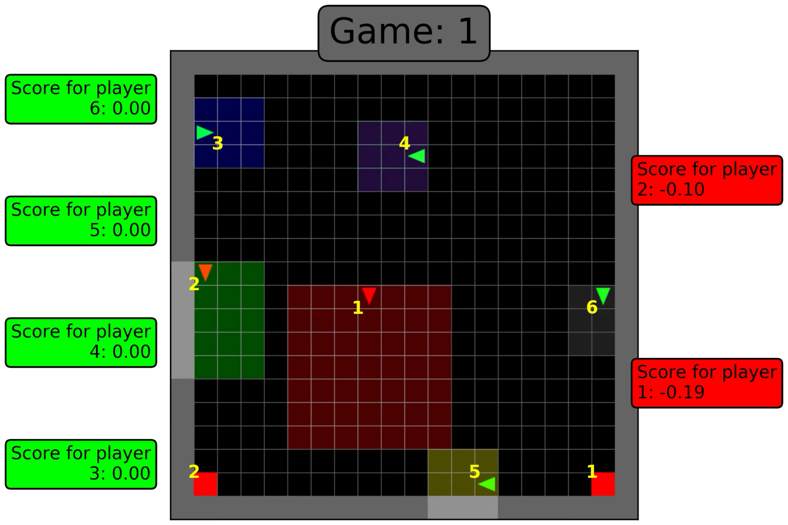


Figure 3.2: $n_H = 2$, $n_C = 4$ in 18×18 grid world

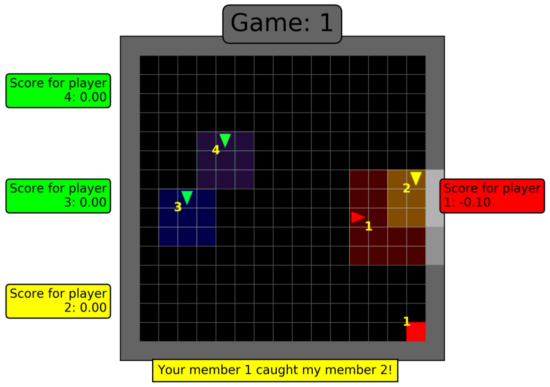


Figure 3.3: $n_H = 1$, $n_C = 3$ in 15×15 grid world

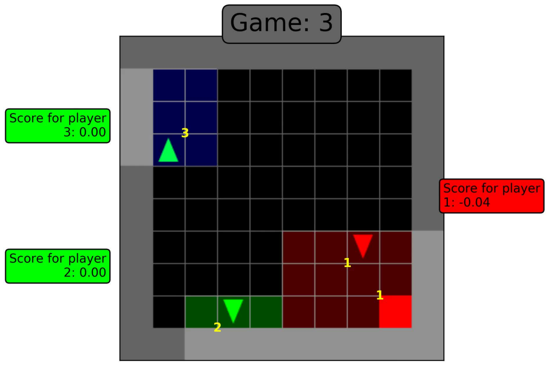


Figure 3.4: $n_H = 1$, $n_C = 2$ in 8×8 grid world

fixed (but unknown to the former team). We can then turn the qualitative rules above into the formulation of a Markov decision problem for the first team. In this report, we describe this process of learning policies for the human team. This process can be repeated for the computer team while keeping the human team policy fixed (though unknown).

The joint human-computer game can be represented by a Markov decision process $\mathcal{M} = \big(\mathcal{S}, \mathcal{A}, \{P_a : a \in \mathcal{A}\}, R, \gamma\big)$ where

1. $\mathcal{S}$ is the state space comprising all possible states of the form

$$s = \big[\{x_i, y_i, \theta_i : i \in [n_H + n_C]\}, \{G_i : i \in [n_H]\}\big] \in \mathbb{R}^{3(n_H+n_C)+2n_H}$$

where we encode the orientation of player i for all $i \in [n_H + n_C]$ as follows:

$$\theta_i = 0 \leftrightarrow \text{East};\ \ \theta_i = 1 \leftrightarrow \text{South};\ \ \theta_i = 2 \leftrightarrow \text{West};\ \ \theta_i = 3 \leftrightarrow \text{North}$$

and the goal of human player $i \in [n_H]$ is denoted by $G_i = (x_{G_i}, y_{G_i})$. One can interpret the above state s as an effective state since it is a concatenation of agent states (here, the human team states) along with the environment states (here the computer team states and its own goals).[3] Since $x_i \in [w]$, $y_i \in [h]$ and $\theta_i \in \{0,1,2,3\}$, we have the dimension of state space as $n = |\mathcal{S}| = 4^{n_C+n_H} \times (hw)^{n_C+2n_H}$.

2. $\mathcal{A} = \mathcal{A}_H$ is the action space for the human team with actions

$$a = \begin{bmatrix} a_1 & a_2 & \cdots & a_{n_H} \end{bmatrix}$$

where we encode the actions of player i for all $i \in [n_H + n_C]$ as follows:

$$a_i = 0 \leftrightarrow \text{Stay Still};\ \ a_i = 1 \leftrightarrow \text{Turn Left};\ \ a_i = 2 \leftrightarrow \text{Turn Right};\ \ a_i = 3 \leftrightarrow \text{Move ahead}$$

Since $a_i \in \{0,1,2,3\}$ for all $i \in [n_H]$, the dimension of the action space is $m = |\mathcal{A}| = 4^{n_H}$.

3. P_a is the transition model under action a where $P_a(s, s')$ is the probability of transitioning into state $s' \in \mathcal{S}$ upon using action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. Note that there is assumed to exist a (possibly unknown) computer policy $\psi : S \to \mathbb{P}(\mathcal{A}_C)$ that determines the computer action given the effective state. Hence, given the effective state at the current time instant, one can in principle determine the next effective state given the human action.

4. $R(s, a)$ is the reward for taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. This can equivalently be expressed in terms of the random variable $R(s')$ where the reward takes on values $R(s')$ with probability $P_a(s, s')$ for next state s'. To make notation simpler, we define the reward function for the human team as the sum of those got by its players. The reward $R_i(s')$ got by player $i \in [n_H]$ is given by

$$R_i(s') = \begin{cases} R, \text{ if player } i \text{ reaches its goal, when in state } s' \\ \text{R+LC}, \begin{array}{l}\text{if player } i \text{ reaches its goal and has captured } L \text{ computer players} \\ \text{along the way, when in state } s'\end{array} \\ -C, \text{ if player } i \text{ is captured by a computer player, when in state } s' \\ -\lambda_1\big(|x_i - x_{G_i}| + |y_i - y_{G_i}|\big) - \lambda_2, \text{ otherwise} \end{cases}$$

[3]In experiments, we augment the state s with $\big[\{D(i, G_i) : i \in [n_H]\}, \{D(i, j) : i \in [n_H]; j \in n_H + [n_C]\}\big]$ where $D(i, j) = |x_i - x_j| + |y_i - y_j|$ for $i \in [n_H]; j \in n_H + [n_C]$ is the l_1 distance between human player i and computer player j and $D(i, G_i) := |x_i - x_{G_i}| + |y_i - y_{G_i}|$ is the l_1 distance of human player $i \in [n_H]$ from its own goal $G_i = (x_{G_i}, y_{G_i})$. This was done since it was observed to improve performance.

where $\lambda_1, \lambda_2 \geq 0$ are weighting factors for the cost of being at a certain distance to the goal, and the cost of maneuver time respectively. The reward for the human team is then given by

$$R(s') = \sum_{i=1}^{n_H} R_i(s')$$

Note that is the reward function that is used to generate expert trajectories for imitation learning algorithms. It can also be used (as in this report) to compare the performance of reinforcement learning algorithms to imitation learning algorithms for this problem.

5. $\gamma \in [0, 1)$ is the discount factor.

Since the dimension of the state and action spaces are both exponential in the number of players n_C and n_H, function approximators are essential to reducing the computational complexity of learning algorithms on these spaces. We will therefore use multi-layer perceptron networks to compute functions of interest on $\mathcal{S}$ and $\mathcal{A}$. With the above MDP formulation, we now introduce an imitation learning algorithm of interest called generative adversarial imitation learning.

3.3 Generative Adversarial Imitation Learning (GAIL)

The authors of GAIL [9] are interested in the problem of learning to perform a task from expert demonstrations, which is a specific setting of imitation learning. They introduce a framework for directly learning policies from data, bypassing any intermediate IRL step by looking at the policy given by running reinforcement learning on a cost function learned by maximum causal entropy IRL [25, 24]. In order to build up to their framework, we first look at the cost function output by a maximum entropy IRL procedure on an expert policy $\pi_E \in \Pi$. Denote by $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ the set of all cost functions $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Definition 8 (Cost from MaxEntIRL)**.** *The output of running maximum entropy IRL with expert policy $\pi_E \in \Pi$ and convex cost regularizer $\psi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R} \cup \{\infty\}$ is a cost function denoted by $\mathrm{IRL}_\psi(\pi_E)$ and defined as*

$$\mathrm{IRL}_\psi(\pi_E) := \arg\max_{c \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}} -\psi(c) + \left[\min_{\pi\in\Pi} \mathbb{E}_\pi[c(s,a)] - H(\pi)\right] - \mathbb{E}_{\pi_E}[c(s,a)] \tag{3.1}$$

Note that (3.1) is the same as (2.7) with a cost regularizer ψ. Given the cost function that is output by IRL, we are now interested in the policy given by running reinforcement learning on the output of IRL. In order to characterize the optimal policy from RL, the authors transform optimization problems over policies into convex problems by mapping a policy $\pi \in \Pi$ to its occupancy measure ρ_π as given in (2.2).

Proposition 1 ([19])**.** *The set of all possible occupancy measures $\mathcal{D} = \{\rho_\pi : \pi \in \Pi\}$ has a one-to-one correspondence to the set of all policies Π. In addition, $\mathcal{D}$ is a convex set since it can be expressed as $\mathcal{D} = \big\{\rho \geq 0 : \sum_a \rho(s,a) = p_0(s) + \gamma \sum_{s',a} P(s|s',a)\rho(s',a)\ \forall s \in \mathcal{S}\big\}$ where p_0 is the initial state distribution.*

One can then characterize the policy got by running reinforcement learning on the cost output by maximum entropy inverse reinforcement learning as follows.

Proposition 2 ([9])**.**

$$\mathrm{RL} \circ \mathrm{IRL}_\psi(\pi_E) = \arg\min_{\pi \in \Pi} -H(\pi) + \psi^\star\big(\rho_\pi - \rho_{\pi_E}\big) \tag{3.2}$$

$$= \arg\min_{\pi \in \Pi} \max_{c \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}} -H(\pi) + (\rho_\pi - \rho_{\pi_E})^T c - \psi(c) \tag{3.3}$$

where $\psi^\star : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R} \cup \{\infty\}$ is the convex conjugate function of the convex cost regularizer $\psi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R} \cup \{\infty\}$.

The proof of Proposition 2 is in the Appendix of [9]. (3.3) says that the optimal cost function and optimal policy form a saddle point of a certain function. IRL finds the optimal cost function and running RL on the output of IRL gives the optimal policy. (3.2) says that ψ-regularized IRL implicitly seeks a policy whose occupancy measure is close in the sense of $\psi^\star$ to that of the expert. Hence, the authors make the following interesting statement: *While IRL was originally defined as the problem of finding a cost function that makes the expert policy optimal with respect to derived cost, Proposition 2 helps us view IRL as the problem of finding a cost function that induces a policy that matches the expert's occupancy measure.*

An interesting exercise is to analyze the result of using different cost regularizers ψ in Proposition 2 to see if we can derive some existing imitation learning algorithms as special cases of the analysis above. Table 3.1 shows the results from such an analysis for two cost functions that give rise to existing imitation learning algorithms, as well a novel algorithm that connects to the area of generative adversarial networks [8].

Consider the following cost regularizer, called the GAIL regularizer, given by

$$\psi_{\mathrm{GAIL}}(c) := \begin{cases} \mathbb{E}_{\pi_E}[g(c(s,a)], \text{ if } c < 0 \\ +\infty, \text{ otherwise} \end{cases} \quad \text{where } g(x) := \begin{cases} -x - \log(1-e^x), \text{ if } x < 0 \\ +\infty, \text{ otherwise} \end{cases}$$

$$\Rightarrow \psi_{\mathrm{GAIL}}(c) = \begin{cases} -\mathbb{E}_{\pi_E}[c(s,a)] - \mathbb{E}_{\pi_E}[\log(1-e^{c(s,a)})], \text{ if } c < 0 \\ +\infty, \text{ otherwise} \end{cases}$$

Define the function $D : \mathcal{S} \times \mathcal{A} \to [0,\infty)$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$ as

$$D(s,a) := e^{c(s,a)}$$

so that $c(s,a) = \log D(s,a)$. We can then express the GAIL regularizer in terms of D as follows

$$\psi_{\mathrm{GAIL}}(D) = \begin{cases} -\mathbb{E}_{\pi_E}[\log D(s,a)] - \mathbb{E}_{\pi_E}[\log(1-D(s,a))], \text{ if } D \in (0,1) \\ +\infty, \text{ otherwise} \end{cases}$$

$\psi(c)$	Result	Algorithm obtained
Constant function $\psi(c) = k \ \forall c$	$\rho_{\mathrm{RL} \circ \mathrm{IRL}_{\psi}(\pi_E)} = \rho_{\pi_E}$	Behavior Cloning
Indicator function $\psi(c) = \begin{cases} 0, \text{ if } c \in \mathcal{C} \\ +\infty, \text{ otherwise} \end{cases}$	$\psi^{\star}(\rho_\pi - \rho_{\pi_E}) = \max_{c \in \mathcal{C}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)]$	Entropy-regularized apprenticeship learning
GAIL regularizer $\psi(c) = \begin{cases} \mathbb{E}_{\pi_E}[g(c(s,a)], \text{ if } c < 0 \\ +\infty, \text{ otherwise} \end{cases}$	$\psi^{\star}(\rho_\pi - \rho_{\pi_E}) = D_{\mathrm{JS}}\big(\rho_\pi \big\|\rho_{\pi_E}\big) + k$	Generative adversarial imitation learning (GAIL)

Table 3.1: Imitation learning algorithms as special cases of Proposition 2

Proposition 3 ([9])**.** *The convex conjugate of ψ_{GAIL} satisfies*

$$\begin{aligned}\psi^{\star}_{\mathrm{GAIL}}(\rho_\pi - \rho_{\pi_E}) &= \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi[\log D(s,a)] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] \\ &= D_{\mathrm{JS}}\big(\rho_\pi \big\|\rho_{\pi_E}\big) + k\end{aligned}$$

where $D_{\mathrm{JS}}(P||Q)$ denotes the Jensen-Shannon divergence between probability distributions P and Q, and $k \in \mathbb{R}$ is a constant.

Using Propositions 2 and 3, we can see that

$$\mathrm{RL} \circ \mathrm{IRL}_{\psi_{\mathrm{GAIL}}}(\pi_E) = \arg\min_{\pi} -H(\pi) + D_{\mathrm{JS}}\big(\rho_\pi \big\|\rho_{\pi_E}\big) \tag{3.4}$$

This says that the GAIL objective is to find a policy that has occupancy measure *close* to that of the expert where closeness is defined in terms off the Jensen-Shannon divergence between distributions, while having high entropy. (3.4) can equivalently be expressed as

$$\boxed{\mathrm{RL} \circ \mathrm{IRL}_{\psi_{\mathrm{GAIL}}}(\pi_E) = \arg\min_{\pi} \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi[\log D(s,a)] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - H(\pi)} \tag{3.5}$$

which is similar to the objective in generative adversarial networks [8] given below

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}\big[\log D(x)\big] + \mathbb{E}_{z \sim p_Z(z)}\big[\log(1 - D(G(z)))\big]$$

which represents a two player min max game between a generator G and a discriminator D. The aim of the generator is to learn to generate samples from the distribution of the training data p_{data}. And, the aim of the discriminator is to differentiate the distribution of the generated data $G(z)$ from the training data distribution p_{data}. Hence, G tries to *fool* D by minimizing the probability that D classifies the generated data as not coming from p_{data} as D tries to prevent any such activity by G. In (3.5), the policy π plays the role of the generator that tries to mimic the expert occupancy measure ρ_{π_E} while the cost function $D = e^c$ plays the role of the discriminator that tries to differentiate the occupancy measure of the generator ρ_π from that of the expert ρ_{π_E}. This similarity between the optimization problems in generative adversarial networks and GAIL gives GAIL its name.

We now look at how (3.5) is implemented in practice when the generator (policy) and discriminator (cost) are modeled by function approximators. Let θ be the weights of the parameterized policy π_θ and w be the weights of the parameterized cost D_w. A natural approach is to take a gradient ascent step for the weights w and a gradient descent step for the weights θ. Define

$$L(w,\theta,\pi_E) := \mathbb{E}_{\pi_\theta}[\log D_w(s,a)] + \mathbb{E}_{\pi_E}[\log(1-D_w(s,a))] - H(\pi_\theta)$$

The gradient of L with respect to parameters w and θ are then given by

$$\begin{aligned}
\nabla_w L(w,\theta,\pi_E) &= \mathbb{E}_{\pi_\theta}[\nabla_w \log D_w(s,a)] + \mathbb{E}_{\pi_E}[\nabla_w \log(1-D_w(s,a))] \qquad (3.6)\\
\nabla_\theta L(w,\theta,\pi_E) &= -\nabla_\theta H(\pi_\theta) + \nabla_\theta \mathbb{E}_{\pi_\theta}[\log D_w(s,a)]\\
&= -\nabla_\theta H(\pi_\theta) + \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \log D_w(s_t,a_t)\middle| a_t\sim\pi_\theta(\cdot|s_t), s_{t+1}\sim P(\cdot|s_t,a_t)\right]\\
&= -\nabla_\theta H(\pi_\theta) + \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_{a_t\in\mathcal{A}} \log D_w(s_t,a_t)\pi_\theta(a_t|s_t)\middle| s_{t+1}\sim P(\cdot|s_t,a_t)\right]\\
&= -\nabla_\theta H(\pi_\theta) + \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \sum_{a_t\in\mathcal{A}} \log D_w(s_t,a_t)\nabla_\theta\pi_\theta(a_t|s_t)\middle| s_{t+1}\sim P(\cdot|s_t,a_t)\right]\\
&= -\nabla_\theta H(\pi_\theta) + \sum_{t=0}^{\infty}\gamma^t \sum_{a_t\in\mathcal{A}} \mathbb{E}\left[\log D_w(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\pi_\theta(a_t|s_t)\middle| s_{t+1}\sim P(\cdot|s_t,a_t)\right]\\
\nabla_\theta L(w,\theta,\pi_E) &= -\nabla_\theta H(\pi_\theta) + \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log\pi_\theta(a|s) Q(s,a)\right]
\end{aligned}$$

where $Q(s,a) := \mathbb{E}_{\pi_\theta}\big[\log D_w(s',a')\big|s_0 = s, a_0 = a\big]$. Given finitely many trajectories $\tau\sim\pi_\theta$ and $\tau_E\sim\pi_E$, we replace the population averages by their sample averages to form unbiased estimates of the gradients. This gives us algorithm 1 that describes the implementation of generative adversarial imitation learning with function approximation for the cost function D and policy π. $\{\alpha_i : i\geq 0\}$ and $\{\beta_i : i \geq 0\}$ are the learning rates for the discriminator network and generator network respectively.

Algorithm 1: Generative adversarial imitation learning

```
  Input: Expert trajectories $\tau_E \sim \pi_E$, initial parameters $\theta_0$ and $w_0$
1 for $i = 0, 1, 2, \cdots$ do
2     Sample trajectories $\tau_i \sim \pi_{\theta_i}$
3     Gradient ascent for discriminator (cost function) parameters
        $w_{i+1} \leftarrow w_i + \alpha_i \Big[ \hat{\mathbb{E}}_{\tau_i}\big[\nabla_w \log D_w(s,a)\big] + \hat{\mathbb{E}}_{\tau_E}\big[\nabla_w \log(1 - D_w(s,a))\big] \Big]_{w=w_i}$
4     Gradient descent for generator (policy function) parameters
        $\theta_{i+1} \leftarrow \theta_i - \beta_i \Big[ \hat{\mathbb{E}}_{\tau_i}\big[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\big] - \nabla_\theta H(\pi_\theta) \Big]_{\theta=\theta_i}$ where
        $Q(s,a) = \hat{\mathbb{E}}_{\tau_i}\big[\log D_{w_{i+1}}(s', a') \big| s_0 = s, a_0 = a\big]$
5 end
```

Chapter 4

Experimental Results

In this chapter, we investigate the training time and performance of state of the art reinforcement learning algorithms as well as imitation learning algorithms on a particular instance of the game described in Chapter 3.[1] While the general formulation of the MDP and the code can handle arbitrary games, we examine the performance of the different algorithms on a game setup with the following specifications:

n_H	n_C	w	h	v_H	v_C	Capture region H	Capture region C	Goal H	R	C	λ_1	λ_2
1	2	8	8	1	$\{3,5\}$	5×5	$\{3\times 3, 3\times 3\}$	$(8,8)$	1	1	0.01	0.01

As per the above game setup setup, we have 2 computer players and 1 human player in a grid world. The human player has a goal that it needs to get to while avoiding capture by any computer player. The goal of the computer team is to capture the human player before it gets to its goal. As mentioned in the problem setup, the computer team has a speed advantage but capture region disadvantage as compared to the human team. Figure 4.1 gives snapshots of the game window for the above game setup. In the figure on the left, the computer player numbered 2 has captured the human player in its location (and is hence, not visible). In the figure on the right, the human player in red has captured the computer player numbered 2, making it yellow. The capture region of the human player is a square of size 5 while those of both computer players are squares of size 3. The red square on the bottom right of both grids represents the goal for the human player. The strategy of the computer team is for each of its players to find the closest human player to itself, and take the shortest path to that human player. This strategy of the computer team is unknown to the human team (as expected).

[1]All code for this project can be found at `https://github.com/KshamaDw/collaborative-competitive-games`. The grid world setup for collaborative-competitive games is a custom Gym environment [4] written by me that utilizes features from the grid world implementation of [7].

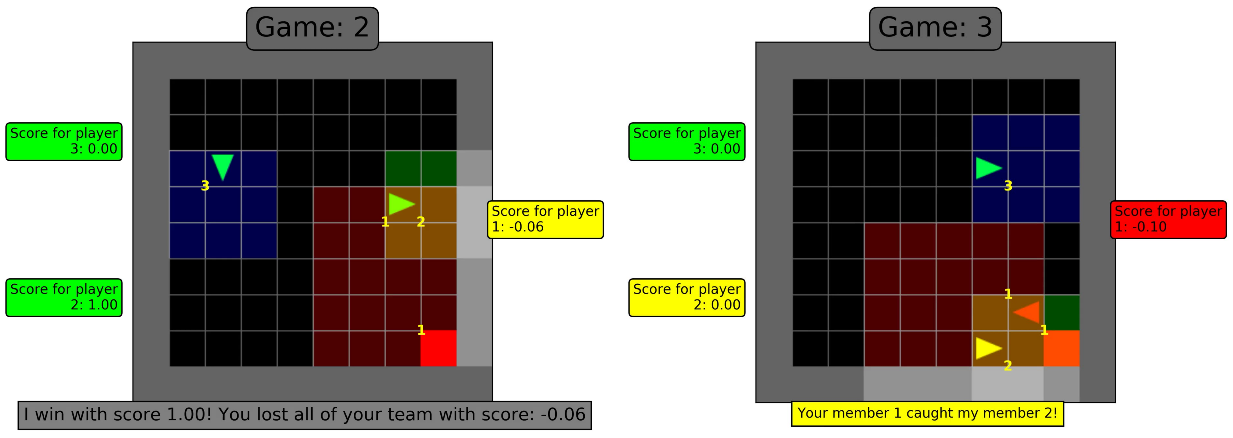

Game: 2
Score for player 3: 0.00
Score for player 2: 1.00
Score for player 1: -0.06
I win with score 1.00! You lost all of your team with score: -0.06
Game: 3
Score for player 3: 0.00
Score for player 2: 0.00
Score for player 1: -0.10
Your member 1 caught my member 2!

Figure 4.1: Game Setup

4.1 Expert data

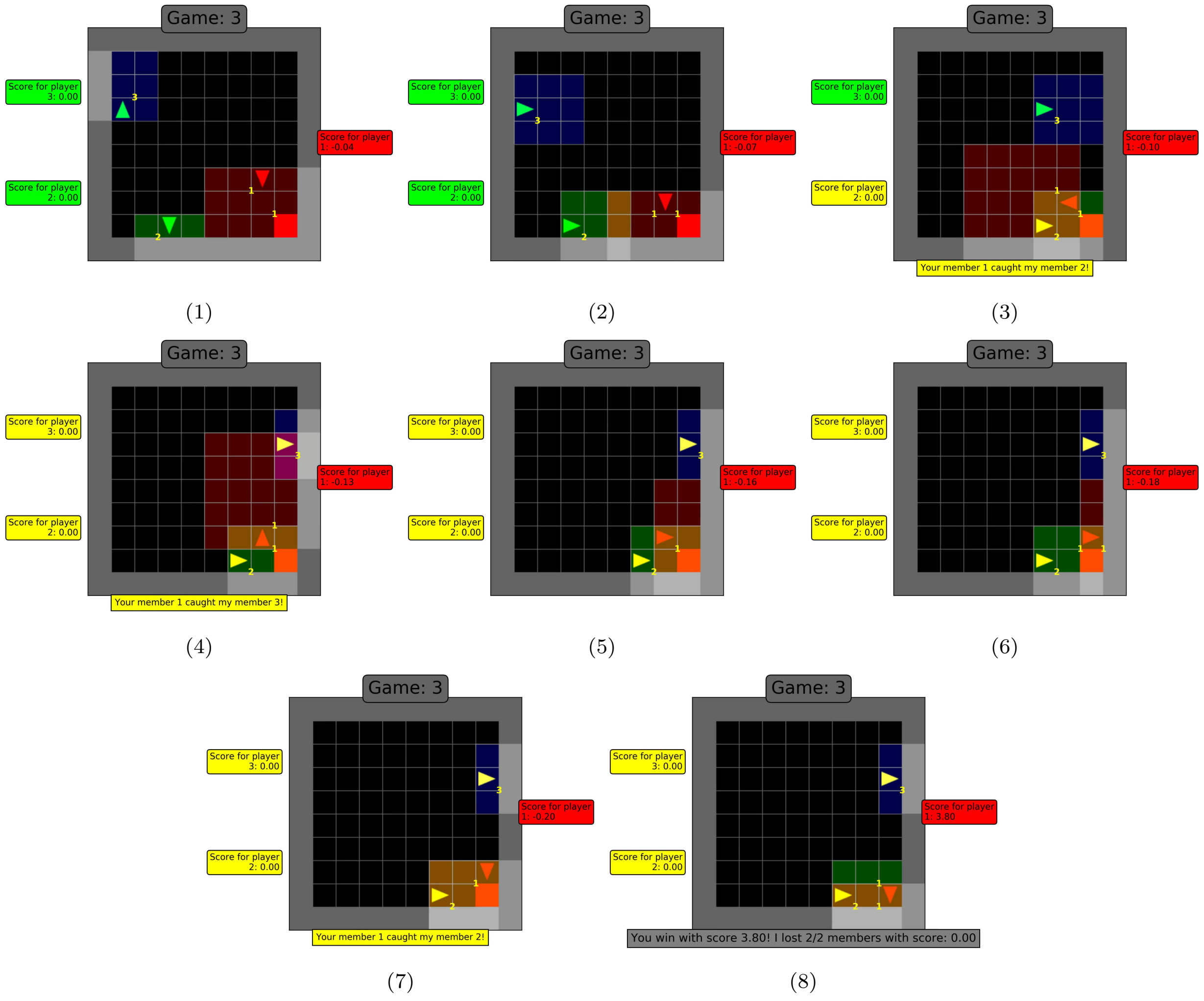
In this project, we consider the following two types of expert data for imitation learning:

- Human game play data: This data was collected from games played by a few friends of mine. This involves the human looking at the grid world with player scores and taking actions to control the human player to increase his/her displayed score. An example sequence of human expert actions is given in Figure 4.2. We found that 100 such games take about 10 minutes to be played. Due to human time considerations, this dataset is limited in size to about 10^4 games. Upon examining the test accuracy of the studied imitation learning algorithms, it was clear that larger expert datasets would help improve the performance of the algorithms. Here, we measure performance in terms of the number of games won of a 1000 new games played.

- RL generated expert game play data: This is got by using the reinforcement learning algorithm of trust region policy optimization (TRPO) [17] using the reward function described in the MDP formulation in Chapter 3. This helps generate large datasets containing up to 10^7 games on a personal computer.[2]

Since finding large quantities of human expert data is hard, we used TRPO generated expert data to train imitation learning algorithms. Since the reward function is also available from the problem formulation, we compare the performance of direct reinforcement learning to that of imitation learning on the human generated as well as TRPO generated data.

[2]All experiments in this report were carried out on a MacBook Pro with 2.7 GHz Quad-Core Intel Core i7 with 16GB RAM and Intel Iris Plus Graphics 655 1536 MB.



Your member 1 caught my member 2!

Your member 1 caught my member 3!

Your member 1 caught my member 2!

You win with score 3.80! I lost 2/2 members with score: 0.00

Figure 4.2: Example of human expert game play

4.2 Reinforcement Learning algorithms

Reinforcement learning algorithms seek to find an optimal policy for a system given its MDP formulation through trial-and-error based interactions with the system. Based on their approach towards policy optimization, they can be divided into two types - policy iteration methods and policy gradient methods. Policy iteration methods involve an alternation between value estimation and policy update [3]. And, policy gradient methods involve policy updates using the gradient of the cost function with respect to the policy. In this project, we evaluate the performance of four state of the art reinforcement learning algorithms on their ability to learn to play collaborative-competitive games as described in Chapter 3.

Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) is a policy gradient method that seeks to iteratively update the current policy by minimizing a local approximation to the cost of the policy subject to the constraint that the new policy iterate lies in a trust region around the current policy iterate [17]. Here, the local approximation to the cost of the current policy iterate is got by expressing the expected cost of the new policy in terms of the advantage and state visitation frequency of the current policy as in [10].

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is also a policy gradient method that alternates between interaction with the environment, and optimization of a surrogate objective function using stochastic gradient descent (for cost functions) [16]. There are two main differences between TRPO and PPO. The first difference is the modification of the objective function through the use of a penalty term in PPO instead of a constraint as in TRPO. The second difference is the use of multiple epochs of gradient descent towards each policy update in PPO in order to reduce variance of the updates.

Actor Critic using Kronecker-Factored Trust Region

Actor Critic using Kronecker-Factored Trust Region (ACKTR - pronounced ‘actor’) is a scalable trust-region optimization algorithm for actor-critic methods [23]. Actor-critic methods are policy iteration methods where the temporal difference error from the value function is used to update the policy [18]. The name ‘actor’ refers to the policy that picks actions, and the name ‘critic’ refers to the value function estimator that evaluates the action taken by the actor. Therefore, ACKTR is a policy iteration method that is based on temporal difference learning and trust region optimization, made to be scalable using a Kronecker-factored approximation to the natural gradient [12].

Deep Q Networks

Deep Q Networks (DQN) refer to a modification of the standard Q Learning algorithm [21] through the introduction of non linear features in the representation of the Q value function [13]. The algorithm alternates between updating the Q value based on its deviation from the Bellman equation, and updating the policy to maximize the resulting Q value function. The authors of DQN claim that their algorithm is more data efficient and results in less divergent value functions than those from the standard Q learning algorithm.

We use a neural network with 2 hidden layers with 64 neurons in each layer to model the policy and the value function for all of the four previously mentioned RL algorithms. The nonlinear activation in each layer is the hyperbolic tangent function given by $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The number of training iterations is in the list $\{10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8\}$. The accuracy of the aforementioned RL algorithms is plotted along with the corresponding training time as a function of training iterations in Figure 4.3. We see that PPO achieves the largest test performance by winning about 76% of new games played.

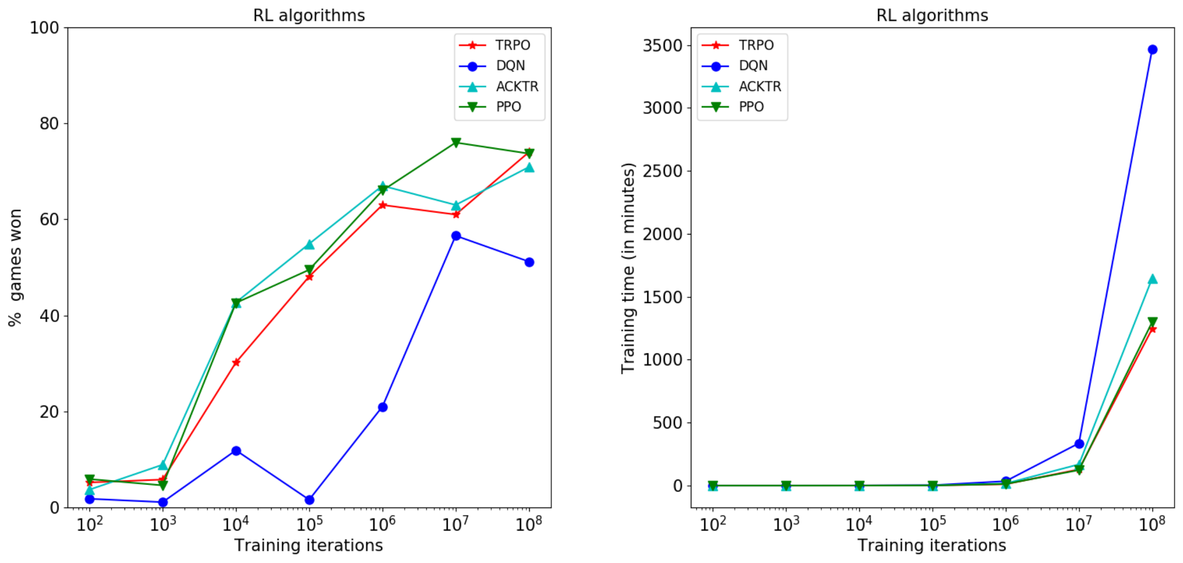


Figure 4.3: RL accuracy and training time

4.3 Imitation learning algorithms

Behavioral Cloning

Behavioral cloning (BC) [2] is a solution method for imitation learning problems wherein training data containing the encountered states and actions of the expert demonstrator is used by the learner or agent to fit a regression model from states to actions to replicate the expert's policy [20]. A major advantage of behavioral cloning is that there is no environment interaction that is needed during training.

We use a neural network with 2 hidden layers with 64 neurons in each layer to model the policy as a function of the state for BC. The nonlinear activation in each layer is the hyperbolic tangent function given by $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The number of training iterations is in the list $\{10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$. The accuracy of BC on using human expert game play data for training is plotted along with the corresponding training time as a function of training iterations in Figure 4.4. Note that the maximum number of human expert games that we could collect was 10^4. As mentioned previously, we also used the RL algorithm of TRPO to automatically generate expert training data for BC. The accuracy of BC on using

TRPO expert data for training is plotted along with the corresponding training time as a function of training iterations in Figure 4.5.[3] In this case, we were able to generate up to 10^7 games on a personal computer using a CPU. The legends in these plots indicate the number of successful games among all expert games, which were then picked out for training BC. The generation times for the human data and TRPO data are plotted as a function of the number of generated games in Figure 4.6. We see that BC achieves a test performance of winning about 67.2% of new games played when trained over 7,145,644 successful games generated by TRPO. Note that best achievable performance when trained with human data is about 59% when trained over 8,126 successful games.

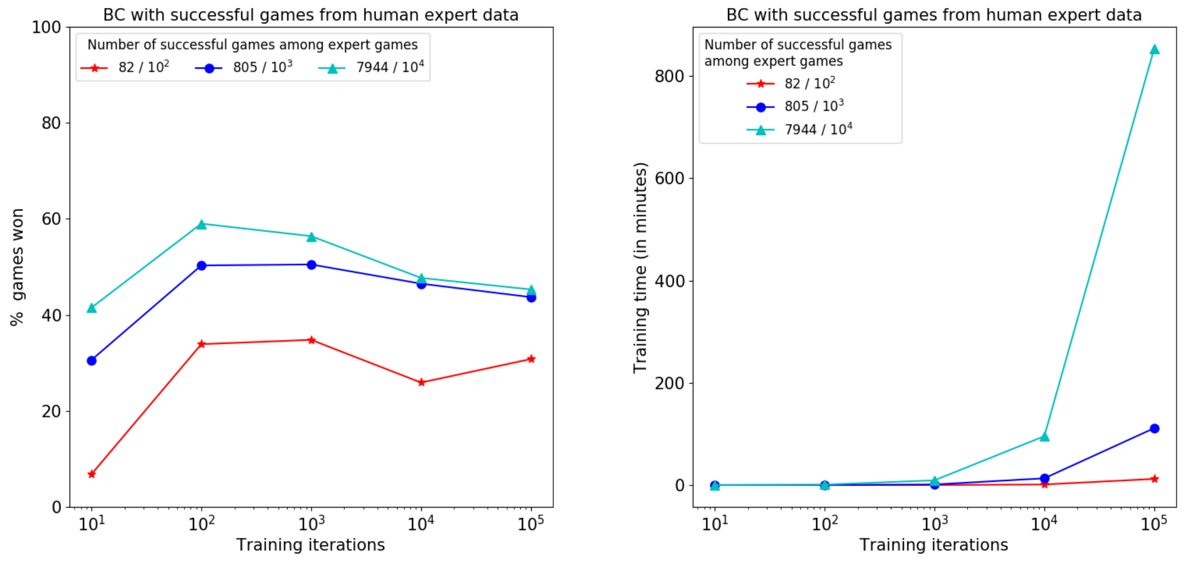


Figure 4.4: BC results on human expert data

Generative adversarial imitation learning

Generative adversarial imitation learning (GAIL) is an approach for imitation learning that was described in detail in the previous chapter. It involves using input expert trajectories to learn a policy that has occupancy measure close to that of the expert (as in Proposition 3), by consecutively learning a cost function that differentiates between the learnt policy and the input expert policy.

[3]All BC experiments carried out on TRPO data were those that completed training within 1 day. There are no results for those that took longer, such as when the data size is 10^7 and training iterations are larger than 10^1.

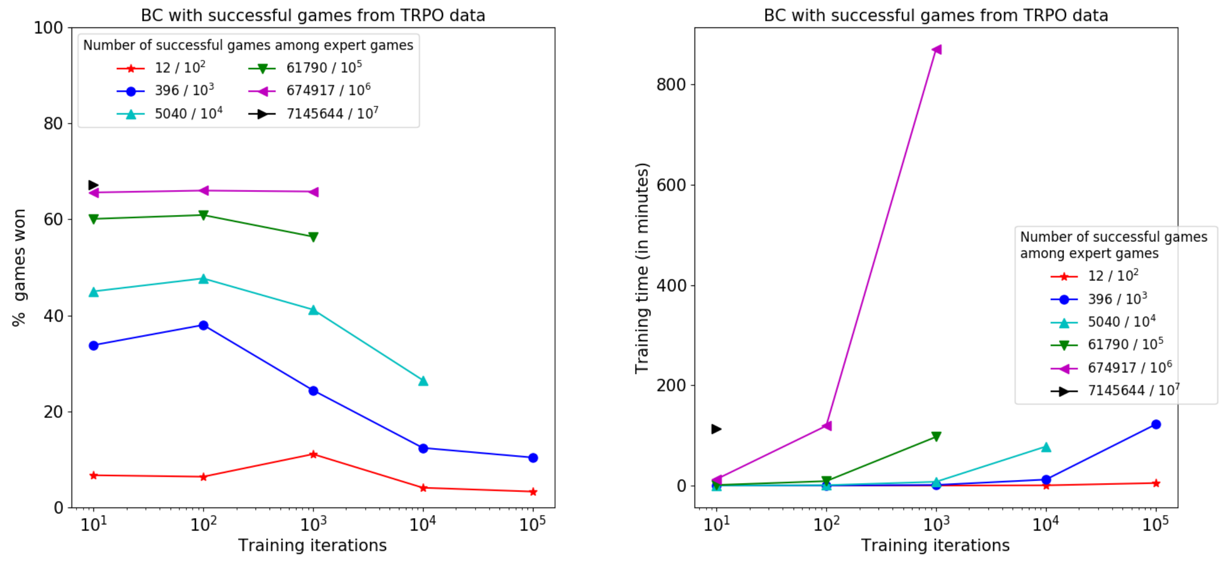


Figure 4.5: BC results on TRPO data

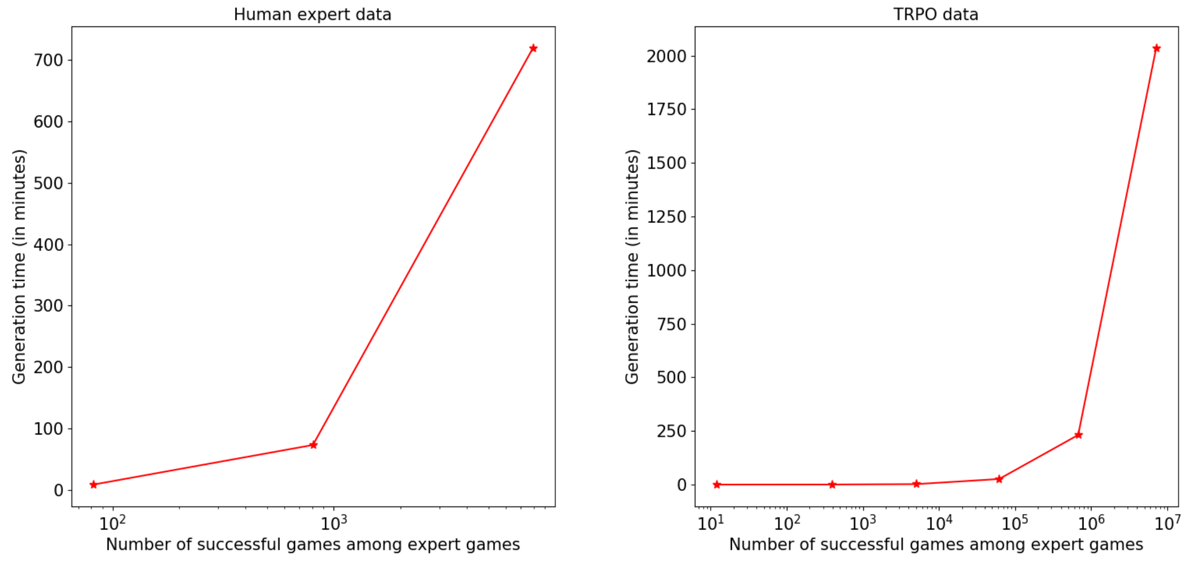


Figure 4.6: Generation time for human expert data and TRPO data

We use a neural network with 2 hidden layers with 64 neurons in each layer to model the policy and the cost function for GAIL. The nonlinear activation in each layer is the hyperbolic tangent function given by $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The number of training iterations is in the list $\{10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$. The accuracy of GAIL on using human expert game play data for training is plotted along with the corresponding training time as a function of training iterations in Figure 4.7. Note that the maximum number of human expert games that we could collect was 10^4. As mentioned previously, we also used the RL algorithm of TRPO to automatically generate expert training data for GAIL. The accuracy of GAIL on using TRPO expert data for training is plotted along with the corresponding training time as a function of training iterations in Figure 4.8. In this case, we were able to generate up to 10^7 games on a personal computer using a CPU. The legends in these plots indicate the number of successful games among all expert games, which were then picked out for training GAIL. The generation times for the human data and TRPO data are plotted as a function of the number of generated games in Figure 4.6. We see that GAIL achieves a test performance of winning about 70.4% of new games played when trained over 7, 145, 644 successful games generated by TRPO. Note that best achievable performance when trained with human data is about 43.4% when trained over 8, 126 successful games.

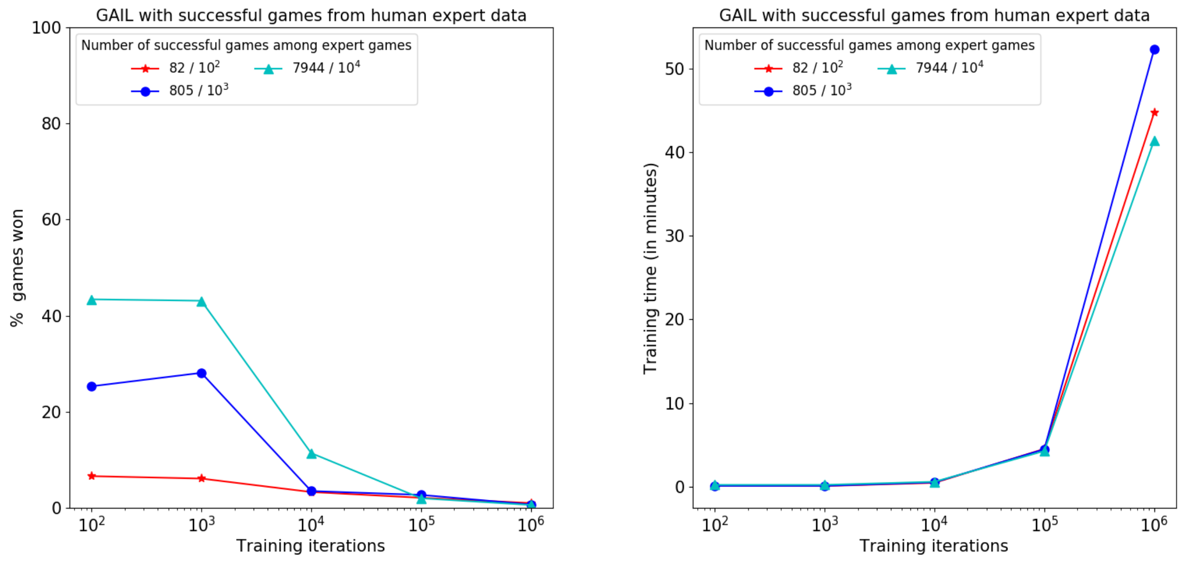
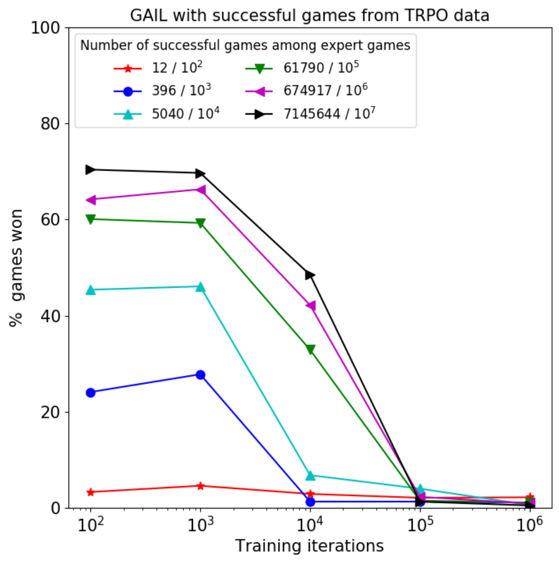


Figure 4.7: GAIL results on human expert data

4.4 Comparison of all methods

In this section we compare the training time and test accuracy of reinforcement learning and imitation learning methods described above for human expert data in Figure 4.9, and for



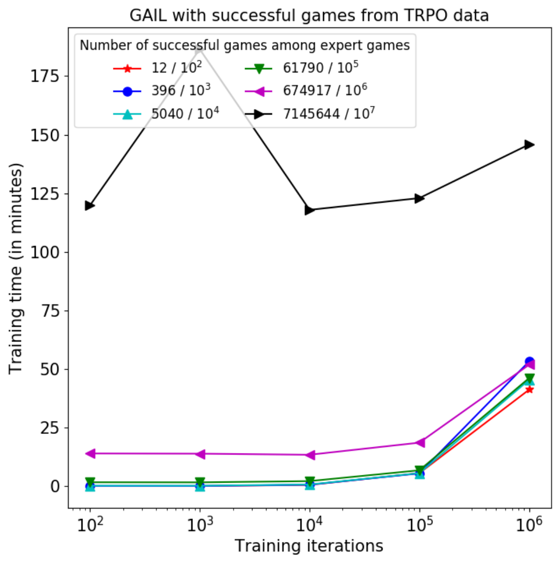


Figure 4.8: GAIL results on TRPO data

TRPO generated expert data in Figure 4.10. We can see that while RL algorithms achieve marginally better test performance than GAIL, they do require the presence of a reward function in order to learn a policy. We also see that GAIL requires lesser training time and has higher accuracy than BC. Some key numbers regarding the best performance of these algorithms on new games are collected in Table 4.1 along with their type - RL refers to reinforcement learning and IL refers to imitation learning.

Algorithm	Type	# Training iterations	# Expert games trained on	Training time (hrs)	% games won
TRPO	RL	10^8	-	20.76	74.0
PPO	RL	10^7	-	2.05	**76.0**
ACKTR	RL	10^8	-	27.44	70.9
DQN	RL	10^7	-	5.61	56.6
BC on human data	IL	10^2	8,126	0.02	59.0
GAIL on human data	IL	10^2	8,126	0.01	43.4
BC on TRPO data	IL	10^1	7,145,644	1.88	67.2
GAIL on TRPO data	IL	10^2	7,145,644	1.99	**70.4**

Table 4.1: Comparison of the performance of all algorithms on new games

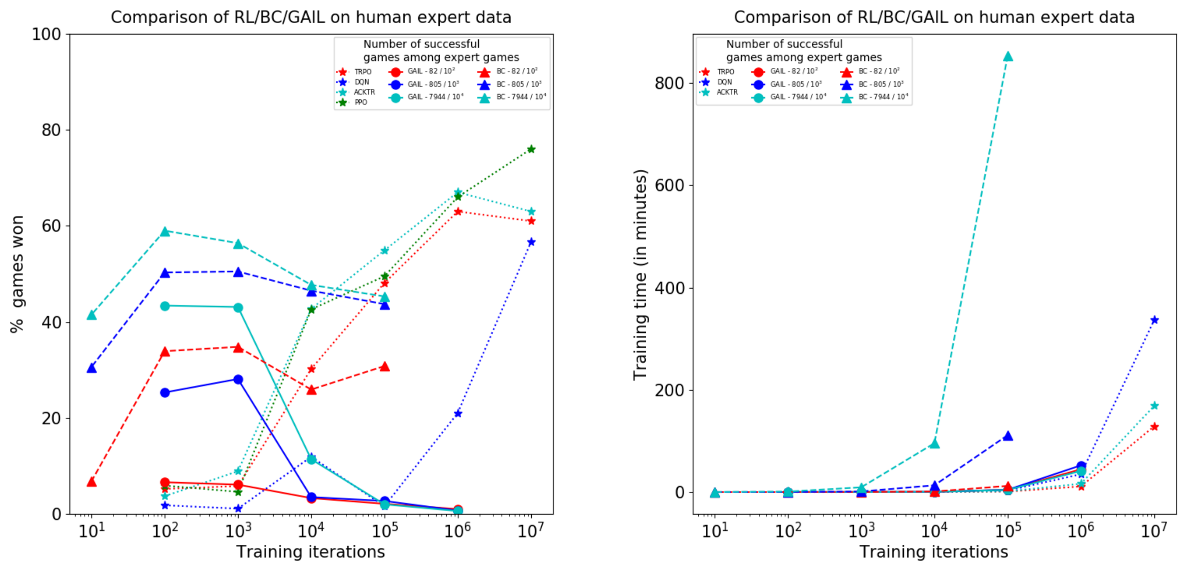


Figure 4.9: Comparison of RL/BC/GAIL on human expert data

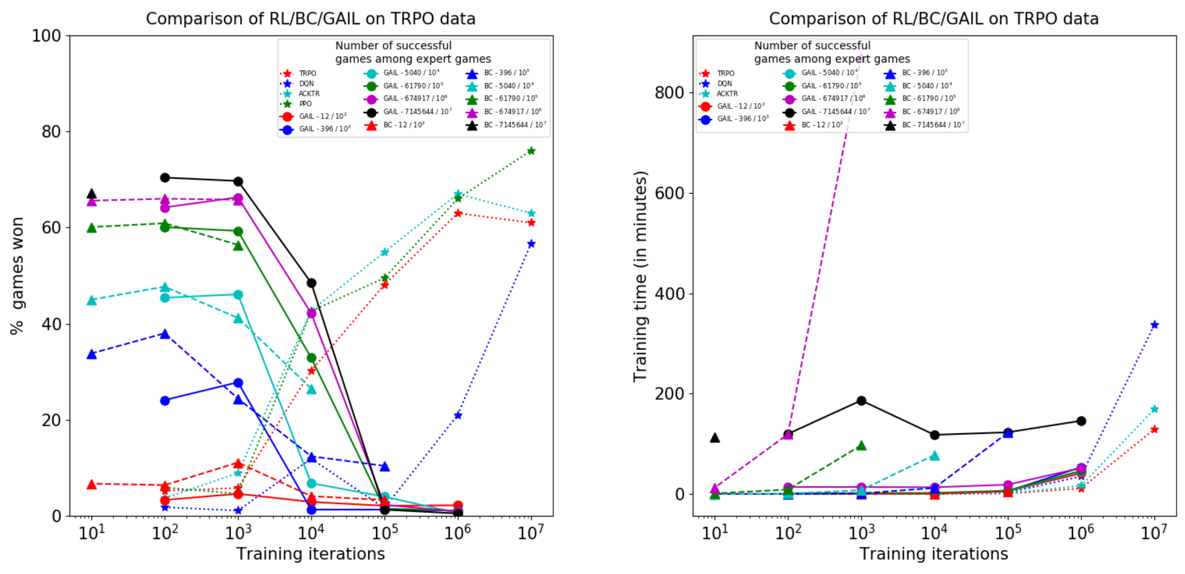


Figure 4.10: Comparison of RL/BC/GAIL on TRPO data

Chapter 5

Discussion and Conclusion

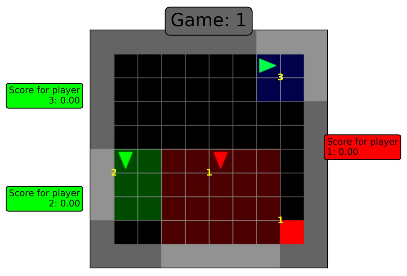
In this chapter, we look at a typical policy learnt from the approaches mentioned in the previous chapter, and contrast it with a typical human expert policy as shown in Figure 4.2. We then collect some results from training GAIL and BC on all expert data, without picking out the successful ones. Some shortcomings of the approach are mentioned thereafter.

5.1 A view of the learnt policies

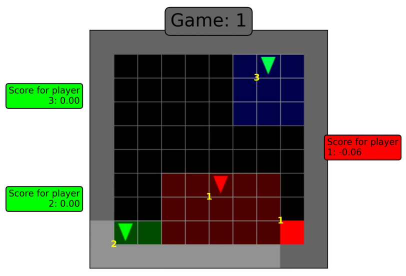
A typical policy learnt from running reinforcement learning using the TRPO algorithm for 10^8 iterations in shown in Figure 5.1. The numbers below the grids indicate the sequence of video frames in the game. The corresponding video can be found at `https://youtu.be/E1sYnvtXcLQ`. A typical policy learnt from running imitation learning using the GAIL algorithm on TRPO generated data of size $7,145,644$ for 100 iterations in shown in Figure 5.2. The numbers below the grids indicate the sequence of video frames in the game. The corresponding video can be found at `https://youtu.be/cEwGMoiMYvc`. One can see that the GAIL player initially starts moving towards its goal in sub-figures (1)-(3). When it realizes that it can increase its reward by also capturing the computer player 2, it changes its orientation to capture it as in sub-figures (4)-(6). It then goes back to its original path of moving towards its goal in sub-figures (7)-(11). It is interesting to see that this reasoning is very similar to that of the human expert policy in Figure 4.2. A similar argument also holds for the RL policy in Figure 5.1.

5.2 Training with *all* data

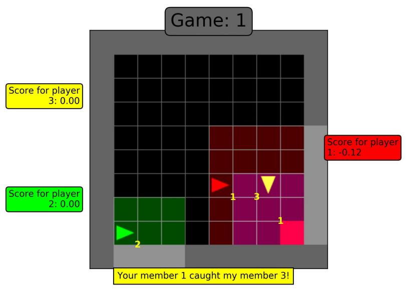
The performance of imitation learning algorithms depends on the quality of the expert data sued for training them. Since the GAIL algorithm is based on maximum entropy formulation of inverse reinforcement learning, it seeks to find a policy that is *similar* to that of the expert subject its justification by the cost function. That is to say that, the learnt policy is free to differ from the expert policy as long as it minimizes the learnt cost function. This hints



(1)

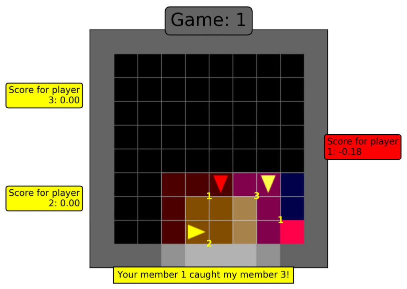


(2)



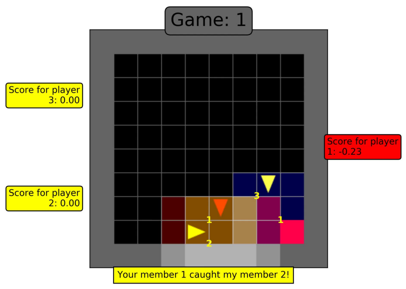
Your member 1 caught my member 3!

(3)



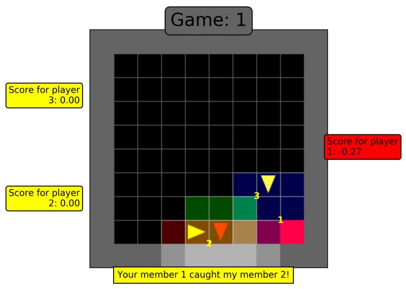
Your member 1 caught my member 3!

(4)



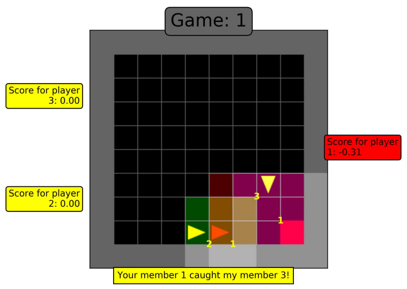
Your member 1 caught my member 2!

(5)



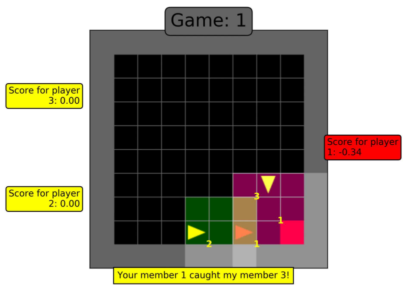
Your member 1 caught my member 2!

(6)



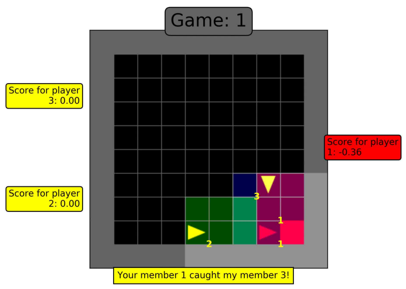
Your member 1 caught my member 3!

(7)



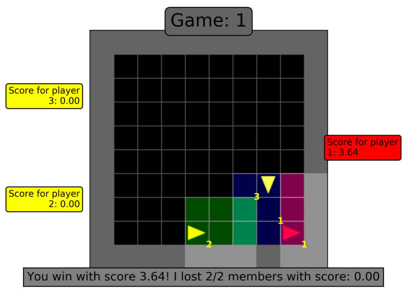
Your member 1 caught my member 3!

(8)



Your member 1 caught my member 3!

(9)

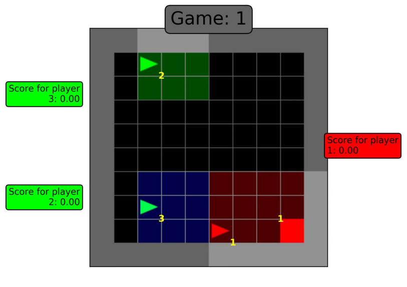


You win with score 3.64! I lost 2/2 members with score: 0.00

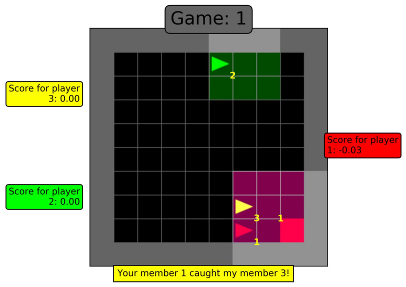
(10)

Figure 5.1: A typical policy learnt using reinforcement learning

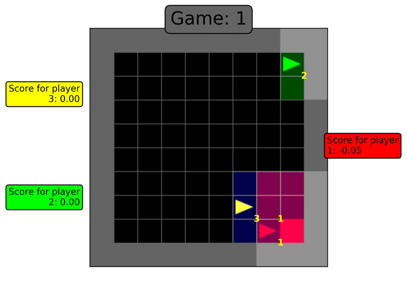
at the possibility that the learnt policy is cost optimizing even when some of the expert trajectories are not from successful games. And that is what we see in our experiments as



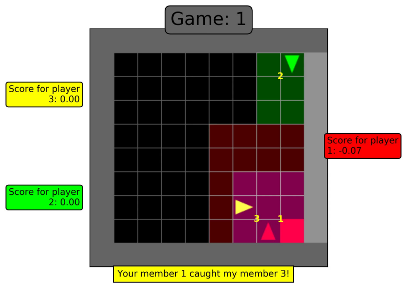
(1)



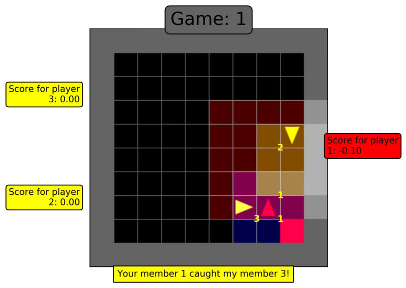
(2)



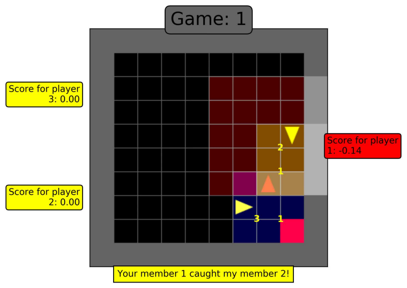
(3)



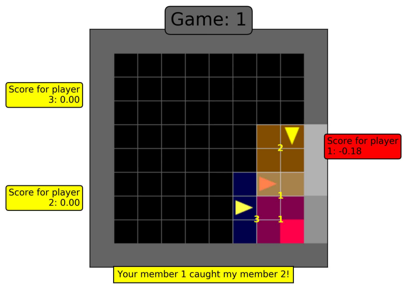
(4)



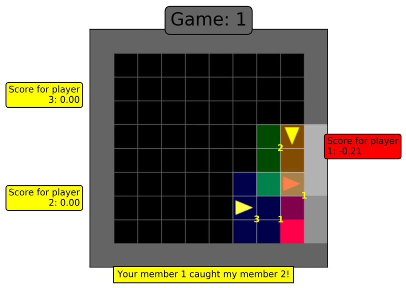
(5)



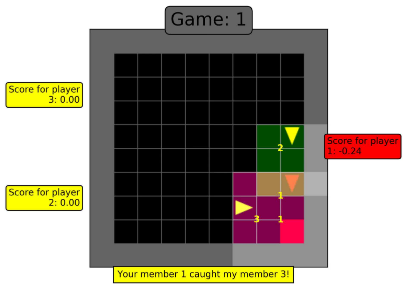
(6)



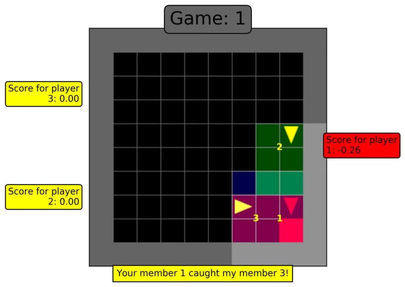
(7)



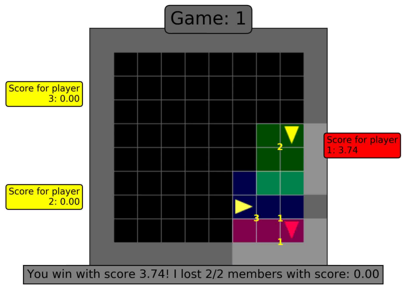
(8)



(9)



(10)



(11)

Figure 5.2: A typical policy learnt using GAIL for imitation learning

well. We see that the performance of GAIL when all of the expert data was used (without picking only the successful samples) is similar to that when only successful samples were used

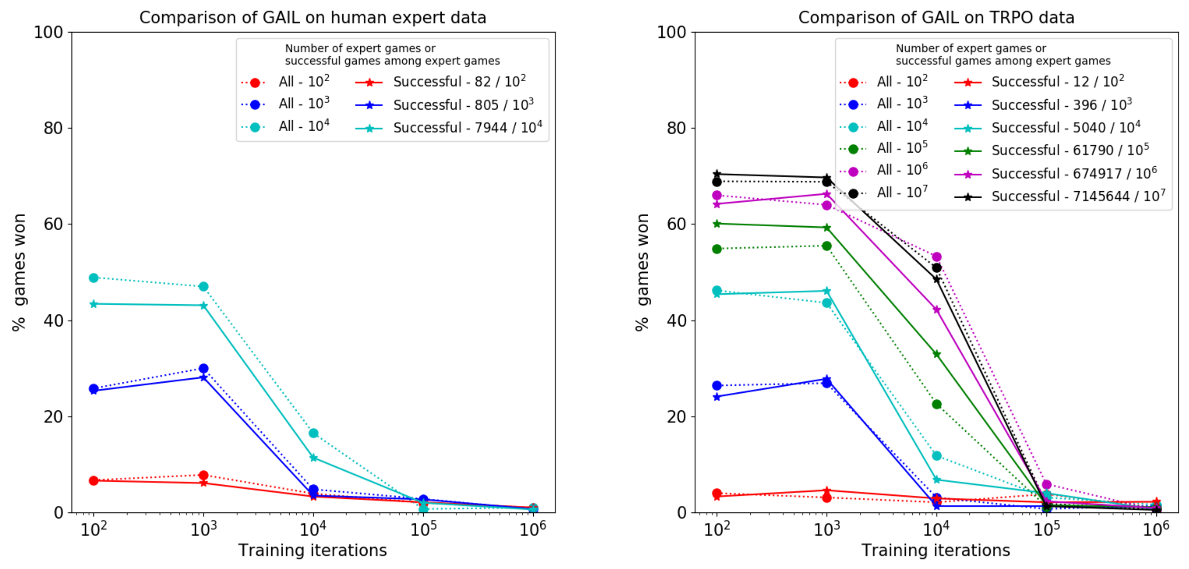


Figure 5.3: Performance of GAIL when trained on all generated data versus that when trained on successful data

for training, as shown in Figure 5.3. An important point to note is that the performance of GAIL is comparable to human expert performance only when the training data is comprised of at least 65% of successful samples.

Conclusion

In this project, we formalized a collaborative-competitive game with competition between two teams and collaboration between members of each team. The ground rules for game play were cast in the form of a Markov Decision Process with the goal of learning optimal game play strategies for members of one team (at a time). We collected expert trajectories from humans that played the game on behalf of one of the teams, and use that data to learn similar game play strategies that can help the team win the game. This involved the examination of an imitation learning approach called Generative Adversarial Imitation Learning that finds a policy *close* to that of the expert, without going through the cycle of IRL followed by RL for imitation learning. We compared the results of running GAIL on expert data to those got from state of the art algorithms from the domain of imitation learning as well as (forward) reinforcement learning using the reward structure described in the problem formulation. We saw that the learnt policies resembled the logic used by humans in playing the game, while being successful in about 70% of new games played. This is very close (though smaller) than the performance of human players. A disadvantage of these imitation learning methods is that they require large quantities of expert data in order to achieve human-like performance. This can be practically infeasible to obtain from human demonstrations, and can benefit from the use of tools such as forward reinforcement learning (when rewards are available) or generative adversarial networks to computationally generate human-like expert data given a training set of human demonstrations.

Bibliography

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

[2] Michael Bain and Claude Sammut. “A Framework for Behavioural Cloning.” In: *Machine Intelligence 15*. 1995, pp. 103–129.

[3] Dimitri P Bertsekas. “Dynamic Programming and Optimal Control, volume Vol. 1”. In: *Athena Scientific, 3rd edition edition* (2005).

[4] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. “A comprehensive survey of multiagent reinforcement learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.

[6] CNBC. *Amazon wins FAA approval for Prime Air drone delivery fleet*. `https://www.cnbc.com/2020/08/31/amazon-prime-now-drone-delivery-fleet-gets-faa-approval.html`. [Online; accessed 03-Nov-2020]. 2020.

[7] Arnaud Fickinger. *Multi-Agent Gridworld Environment for OpenAI Gym*. `https://github.com/ArnaudFickinger/gym-multigrid`. 2020.

[8] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[9] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *arXiv e-prints*, arXiv:1606.03476 (June 2016), arXiv:1606.03476. arXiv: `1606.03476 [cs.LG]`.

[10] Sham Kakade and John Langford. “Approximately optimal approximate reinforcement learning”. In: *ICML*. Vol. 2. 2002, pp. 267–274.

[11] Ryan Lowe et al. “Multi-agent actor-critic for mixed cooperative-competitive environments”. In: *Advances in neural information processing systems*. 2017, pp. 6379–6390.

[12] James Martens and Roger Grosse. “Optimizing neural networks with kronecker-factored approximate curvature”. In: *International conference on machine learning*. 2015, pp. 2408–2417.

[13] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[14] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.” In: *Icml*. Vol. 1. 2000, p. 2.

[15] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. “Maximum margin planning”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 729–736.

[16] John Schulman et al. “Proximal policy optimization algorithms. arXiv 2017”. In: *arXiv preprint arXiv:1707.06347* ().

[17] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. 2015, pp. 1889–1897.

[18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[19] Umar Syed, Michael Bowling, and Robert E Schapire. “Apprenticeship learning using linear programming”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1032–1039.

[20] Faraz Torabi, Garrett Warnell, and Peter Stone. “Behavioral cloning from observation”. In: *arXiv preprint arXiv:1805.01954* (2018).

[21] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[22] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[23] Yuhuai Wu et al. “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation”. In: *Advances in neural information processing systems*. 2017, pp. 5279–5288.

[24] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. “Modeling interaction via the principle of maximum causal entropy”. In: (2010).

[25] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning.” In: 2008.