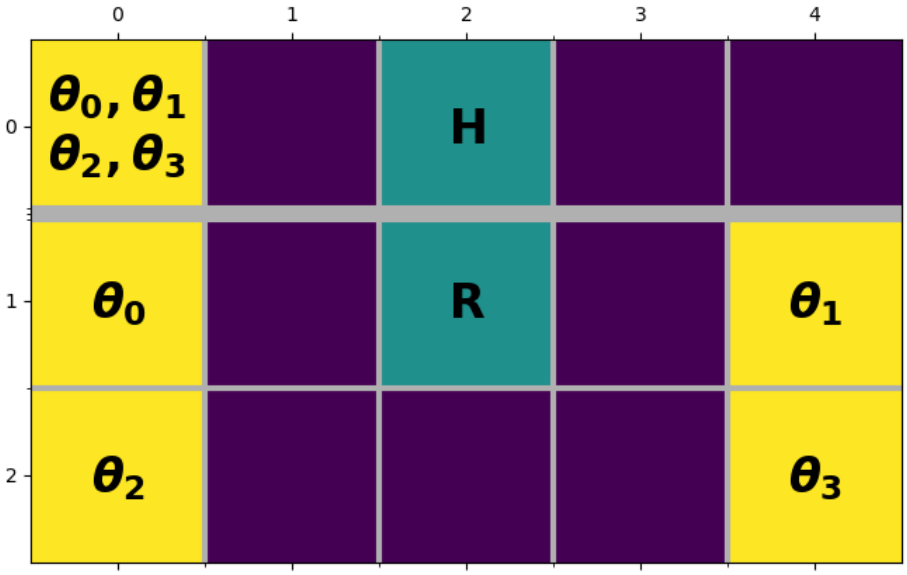


Cooperative Inverse Reinforcement Learning

Cooperation and learning in an asymmetric information setting with a suboptimal teacher

Master's thesis in Complex Adaptive Systems

JOHAN EK

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Cooperative Inverse Reinforcement Learning

Cooperation and learning in an asymmetric information setting with a suboptimal teacher

JOHAN EK

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Cooperative Inverse Reinforcement Learning
Cooperation and learning in an asymmetric information setting with a suboptimal teacher
JOHAN EK

Supervisor: Christos Dimitrakakis, Department of Computer Science and Engineering
Examiner: Carl-Johan Seger, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Rendering of a testing environment for cooperative inverse reinforcement learning. See Section 3.2.1 for details.

Typeset in LaTeX
Gothenburg, Sweden 2018

Cooperative Inverse Reinforcement Learning
Cooperation and learning in an asymmetric information setting with a suboptimal teacher
JOHAN EK
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

There exists many different scenarios where an artificial intelligence (AI) may have to learn from a human. One such scenario is when they both have to cooperate but only the human knows what the goal is. This is the study of cooperative inverse reinforcement learning (CIRL). The purpose of this report is to analyze CIRL when the human is not behaving fully optimally and may make mistakes. The effect of different behaviours by the human is investigated and two frameworks are developed, one for when there is a finite set of possible goals and one for the general case where the set of possible goals is infinite. Two benchmark problems are designed to compare the learning performance. The experiments show that the AI learns, but also that the humans behaviour has a large affect on learning. Also highlighted by the experiments, is the difficulty of differentiating between the actual goal and other possible goals that are similar in some aspects.

Keywords: Machine learning, artificial intelligence, reinforcement learning, cooperation, cirl, computer science, project, thesis.

Acknowledgements

I am grateful to all who have helped me complete this project. Special thanks to Christos Dimitrakakis for teaching me the theoretical background needed for my work and guiding me as my supervisor. Thanks to my examiner Carl-Johan Seger and my peers Hampus Ramström and Johan Backman for providing much appreciated feedback.

Johan Ek, Gothenburg, September 2018

Contents

Acronyms and Nomenclature

AI	Artificial Intelligence
CIRL	Cooperative Inverse Reinforcement Learning
IRL	Inverse Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
LfD	Learning from Demonstrations
RL	Reinforcement learning

*	Short hand notation for optimal or best possible version or option
α	Learning speed
A	A set of actions
a	An action
b	The lowest or highest q-value in some state
δ	The minimum increase in probability for exploration not to be preferred
η	level of optimality for $\mathbf{H}$ or inference strength for $\mathbf{R}$
ϵ	A parameter determining the probability of making a random action
E	The expected value
γ	A discount factor
$\mathbf{H}$	The human agent
κ	A threshold for exploration
ν	Stores the number of times a state has been visited or the number of times each action has been played in a state
n	An integer
ϕ	A probability distribution over Θ constructed by $\mathbf{R}$
π	A policy
P	A probability
P_0	A probability distribution over the initial state s_0 and possibly θ
Q	The expected sum of discounted rewards for some action a in state s
q	A measure of how e□cient an action is at changing the state to some other state
R	A reward function
r	A reward
$\mathbf{R}$	The AI or robotic agent
S	A set of world states
s	A world state
Θ	A set of possible reward parameters
θ	A reward parameter the modifies R
T	A state transition kernel
t	The time step
U	The expected sum of discounted rewards
U	A uniform probability
V	The expected sum of discounted rewards in some state s under the initial distribution of reward parameters and world states

List of Figures

List of Tables

1

Introduction

This chapter introduces the research filed of CIRL and explains the research leading up to it and the connection to other research going on around it. The problem is also made more concrete and the specifics characteristics distinguishing this project from other research is stated along with the aim. An overview of the actual contributions of the project is also presented along with a general outline of the report.

1.1 Background

The research field of artificial intelligence (AI) is an evergrowing area. It has gained massive interest during recent years both from academia and the private sector (A.E.S., 2016). As AI is tasked with solving new kinds of problems, new methods and subfields of research are developed to handle it. Cooperative inverse reinforcement learning (CIRL) is one such new area. It builds upon many layers of earlier research that respectively still are active research topics.

The arguably most active area in artificial intelligence is that of machine learning (ML). It is characterized by the ability of computer programs (AIs) to independently learn from data, i.e. a programmer does not explicitly need to tell the program what to do. ML has an advantage in cases where it is difficult to create explicit algorithms, where a common such case is that of classification. Classification can for example be used to determine whether an email is spam, or for recognizing objects in images.

In classification the AI commonly learns by being presented with a complete training set, i.e. example problems and correct responses. But, in some problems, learning is preferably done by trail and error. This is usually cases that require trade-offs between long-term and short-term rewards (Sutton, Barto, et al., 1998). The subfield reinforcement learning (RL) was developed for this reason and has successfully been applied to board games and simple video games. It was, for example, used by Google DeepMind's Go playing computer program AlphaGo to beat the Go world champion at the time (Shead, 2017).

A problem with RL is that a programmer needs to precisely define a reward function, i.e. a performance criteria of how well the program is behaving. This can in more than one way be a significant disadvantage in certain circumstances. Creating a reward function is simple in problems such as games where a victory equals a high score and a loss equals a low score. However, in a task like driving a car, this is not as simple. There is no winner, instead the right balance between many different safety measures, speed and comfort needs to be found. Most people, even those who can not drive, can see the difference between a good driver and a

bad. At the same time, it is close to impossible to set up a comprehensive set of criteria for differentiating between them.

An incorrectly specified reward function gives rise to the so called Value Alignment Problem (Hadfield-Menell, Russell, Abbeel, & Dragan, 2016). An AI trying to optimize a poorly specified reward function can end up behaving very unpredictable. Russel and Norvig (2010) gives the example of an assistant robot given the task of cleaning up dirt. If it is rewarded every time it cleans up dirt, which may seem reasonable, it will pour it all out again so that it can clean it up repeatedly and thus receive a higher reward. This problem shows up in many places where it is hard to define the reward function. Solving the Value Alignment Problem is crucial to develop safe AIs such as self-driving cars and personal assistant robots.

A method for avoiding these problems with creating a reward function is inverse reinforcement learning (IRL). The basic concept is that instead of providing the AI with a reward function it should learn the reward function as well. The AI infers the reward function by first observing an expert performing the task for some time. This expert is usually a human who is experienced with the task. IRL has the potential to solve the Value Alignment Problem. Since the AI is never completely certain about its objective, it will naturally try to align its learned objective better with that of the expert. IRL has with varying degrees of success been used to train autonomous vehicles (Abbeel, Dolgov, Ng, & Thrun, 2008).

The complexity of the problem can be extended one step further by placing the AI in a cooperative environment together with a human expert. The reasons for cooperation can be that the expert can not complete the task alone, because it is more efficient or since it improves learning in an early phase so that the AI can complete the task alone later. Either way, the AI has to infer the objective and help the expert at the same time. This is problematic since the AI may not be able to learn the objective without helping the expert, and helping is not straightforward when the objective is not known. The AI should not copy the humans behaviour, but choose one that best completes the task together with the human. This may result in a behaviour very different to that of the human. Direct communication between the two agents could be allowed in some situations, but the goal is for the AI to learn as much as possible independently. As an example situation of where CIRL could be useful, imagine that you own a personal assistant robot and both of you are folding laundry. You want the robots help making a three-way fold on a tablecloth. However the robot has never folded anything before. It therefore needs to observe your actions to understand that it is supposed to grab the other end of the fabric and fold it the same way you do.

1.2 Aim

This project aims at aiding in taking the initial steps into the new field of cooperative inverse reinforcement learning. The work on CIRL by Hadfield-Menell et al. (2016) is the platform from which this research starts, but with a major difference in the assumptions. This project is the first that strives to model the human agent as a realistic person, meaning that the human may behave suboptimally and may not understand the reasoning of the AI. How the exact behaviour of the human affects

the learning ability of the AI will be of major interest.

The main goal of the project is to examine whether CIRL is achievable in the general case with a suboptimal human agent. To prove this, the learning mechanism behind a simple general case should be clear, and a simple experiment should show learning in this environment. Or, alternatively, show why it is not possible. In practice, this means designing a simple general CIRL environment as a test-bench and then create a framework for the AI that learns the objective from the human. This framework will be a set of assumptions and algorithms that can be applied to any CIRL problem of a limited complexity.

The difficulty of the problem can be varied over a huge span, from very simple to very hard. The starting point is therefore a simple case to improve understanding, allowing continued work on the more complex cases. The simple case will be studied more rigorously and in greater detail than the more complex problem, which will mainly be a proof of concept. Ideally, the AI should learn the reward fast and understand how to behave to maximize it. This will be tested on the developed test-bench. The algorithms computational complexity will also be discussed. In general, as few results have been published in this area, all new insights are helpful to the continued research.

If the results show that the AI can learn the objective and also how to do it, then it opens up a new way of training AIs. A way that seem very similar to how an animal's parent teaches their children. They are not able to explain what to do, but they can show how to perform the specific task at hand. This way of teaching could be very useful for training robots to perform tasks that humans already can do, or needs to cooperate on. As an example, a personal household assistant robot could be taught a new chore such as brewing coffee, by brewing it together with its owner.

Objectives

- Identify the difficulties of learning in CIRL environments.
- Establish different models of how a human could behave.
- Create learning algorithms for the AI.
- Construct two testing environments for different levels of complexity.
- Evaluate the effect of different assumptions about human behaviour.
- Evaluate the effectiveness of the developed algorithms based on learning speed and robustness.
- Identify optimization opportunities and propose areas of further research.

1.3 Problem

There exists a task and two agents who try to complete it in the best way possible. One agent is a human who knows what the task is and the other is an AI who does not know the task. The AI observes the human to try to understand the task and how to help. The two agents can not communicate in any other way than through their actions in the task environment. Both agents have full knowledge of the environment and the available actions. However, the human is suboptimal

and will therefore make some mistakes. At no point is any external information provided to the AI about the task, and it will therefore never know for sure if it has understood it correctly.

To limit the complexity of the problem, the number of states and actions are assumed to be finite and small. The task is randomly chosen from a set of possible tasks and is therefore not always the same. The set of possible tasks is also known by both agents, as well as the probability of a task being selected. The project is partially divided along the line of complexity. The major difference is whether the set of possible tasks is finite or not. This project builds upon the work by Hadfield-Menell et al. (2016) and generally follows the same formulations, definitions and notations.

1.4 Related work

The work done in this project is inspired by Hadfield-Menell et al. (2016). The main difference is the view of the human agent. While they model the human as an optimal agent, trying to teach the AI in the best possible way, this project does not make such strong assumptions. Instead the human is modeled to make mistakes and be unaware of how to best teach the AI. In their continued work Palaniappan, Malik, Hadfield-Menell, Dragan, and Russell (2017) take the optimal human one step further and develop a fast algorithm for finding the human's optimal teaching behaviour given a specific task. This behaviour requires that the AI assumes that the human is optimal and will provide the AI with the most information possible about the task. This is in stark contrast to the human in this project, which is not assumed to have the interest or ability to act in accordance with an algorithm.

CIRL is an extension to IRL as defined by Abbeel and Ng (2004). IRL also assumes that the human is optimal. Interestingly, Hadfield-Menell et al. (2016) proves that the human can teach the AI more by behaving in a special teaching way that may be a suboptimal policy. In this project, a teaching policy that is also an optimal policy will be tested.

A concept with many similarities to IRL and CIRL is developed by Lin, Beling, and Cogill (2014). It is a Bayesian framework for Multi-agent IRL (MIRL), where two agents then compete in an unknown stochastic environment. Similarly to CIRL, the soundness of each agent's behaviour then heavily depends on the behaviour of the other agent.

Hadfield-Menell et al. (2016) also suggests that CIRL may be a solution to the Value Alignment Problem. Carey (2017) shows that there is no guarantee for the AI to be corrigible, which is necessary to solve the Value Alignment Problem. The AI is only corrigible if it is uncertain about all things that could possibly have an impact on the performance. Incorporating such a broad uncertainty is difficult.

Value Alignment Problems are also studied in other areas such as economics, where they are referred to as principal–agent problems. An example could be an employer trying to incentivise the employees to work in a way that maximizes profit. Specifying the correct incentive is then a problem very similar to specifying a reward function (Jensen & Meckling, 1976).

1.5 Contribution

This report contributes to the CIRL research by taking the initial steps into the area of suboptimal human agents. An analysis of the problem is presented along with two major results.

- For the AI to learn from the human it must make assumptions about the human behaviour. A non-exhaustive list of possible such assumptions is presented and analyzed. These assumptions are then converted to explicit human models and the ability of the AI to learn from these model humans is investigated.
- Two frameworks with learning algorithms are presented for learning in a finite and infinite reward space respectively. Their performances are tested and evaluated on simple custom made example problems.

1.6 Overview

Chapter 2 is a review of the theory necessary for this project and presents mathematical definitions of the problem. Chapter 3 presents new insights and frameworks on how to solve the problem. In Chapter 4 the frameworks are tested experimentally on the developed test-benches. Finally, in Chapter 5, the results of the experiments are discussed and presented as a summary.

2

Theory

This chapter reviews the necessary background theory for CIRL that is needed for the developed frameworks. The problem is also given a solid mathematical definition and explained in great detail. Some other mathematical tools are also presented that will prove useful later.

2.1 Markov Decision Processes

One of the most common ways to model problems or tasks in the field of reinforcement learning is with the mathematical framework Markov Decision Processes (MDPs). It models the problem as a stochastic control process in discrete time. At every time step, the process is in a state and the controller (decision maker) can select any action available at that state. The action causes the process to stochastically move into a new state in the next time step. A reward is also given to the decision maker in each time step. This reward can be either positive or negative and is dependent on the process.

2.1.1 Definition

Definition 1 (Markov Decision Process) *A framework that is characterized by time t and a tuple, $M = S, A, T(\cdot|\cdot,\cdot), R(\cdot,\cdot), P_0(\cdot), \gamma$, with the following definitions:*

S a set of world states: s S.
A a set of actions: a A.
$T(\cdot|\cdot,\cdot)$ a conditional distribution on the next world state, given previous state and action: $T(s_{t+1}|s_t, a_t)$.
$R(\cdot,\cdot)$ a reward function that maps world states and actions to real numbers: $r_t = R(s_t, a_t)$ with r R.
$P_0(\cdot)$ a distribution over the initial state $P_0(s_0)$.
γ a discount factor: γ $[0, 1]$.

The general MDP framework allows state and action space to be either finite or infinite. However, many algorithms require that they are finite. The state transition only depends on the current state and action and is conditionally independent on all previous states and actions. This requirement is called the Markov property. Usually, states can be defined in a way so that this necessary property holds. The discount factor embodies the fact that time matters to the received reward. Without it (identical to $\gamma = 1$) current and future rewards are equally important. There

would then be no incentive to accumulate a reward as long as the possibility to do so in the future does not diminish. On the other hand with $\gamma = 0$ only immediate rewards matter.

2.1.2 Core problem

The point of interest in the MDP is how to chose a behaviour as to maximize the expected discounted sum of rewards

$$U = E\Big[\sum_{t=0}^{\infty} \gamma^t r_t \Big| \pi\Big]. \tag{2.1}$$

Where π is referred to as a policy and defines the behaviour by designating an action in each state, $\pi(s) = a$. The policy or policies that has the highest expected discounted sum of rewards are referred to as optimal policies. A policy can either be deterministic or stochastic.

2.2 Learning from demonstrations

In standard reinforcement learning, the agent explores the environment and tries to adapt its behavior (policy) to maximize the received reward. The reward function R is always part of the MDP, but in some cases it can be hard to specify. Then the agent can not observe the reward and therefore not use it to guide its behavior. However, the agent may be able to learn in other ways. One such way is to substitute the reward with demonstrations of an expert acting in the MDP. The expert is considered to act approximately optimally with regards to the unspecified reward. This is called learning from demonstrations (LfD). The demonstrations can mainly be used in three different ways:

1. **Imitation learning:** The agent simply tries to mimic the behaviour of the expert regardless of whether it is optimal or not.
2. **Inverse reinforcement learning:** The agent only observes and tries to infer the reward function R form the demonstrations.
3. **Apprenticeship learning via inverse reinforcement learning:** The agent first infers the reward function and then tries to optimize its behaviour based on the inferred reward function with regular reinforcement learning.

2.3 Cooperative Inverse Reinforcement Learning

Cooperative Inverse Reinforcement Learning is a modification of standard IRL and apprenticeship IRL. In the control process (MDP) there are now two agents (decision makers) instead of one. They each have a separate set of actions available to them. Their combined action is called the joint action and it controls the process. The difficulty lies in that only one agent knows the objective. The one without that knowledge has to observe the other agents actions and infer the objective. The two agents are usually a human that knows the objective and a robot, or AI, that attempts to learn the objective.

2.3.1 Definition

CIRL still fits in the MDP framework, but with some modifications. The standard action is now split into two parts with separate action spaces. They can both be considered separate actions or as two parts of the same action. Either way they are together called a joint action and it controls the state transitions and the reward. The fact that the reward is unknown by $\mathbf{R}$ is captured by the introduction of the parameter θ that modifies the reward function, see Definition 2. θ is referred to as a reward parameter and can be of any shape and size. In this project it represents either an index or a vector. Recall the example from the Background section with a household assistant robot folding laundry. Imagine that you and the robot are holding on to either side of the tablecloth. The reward parameter θ would then indicate if you want to make a two-way fold, a three-way fold, iron it or any other possible objective you may have. A set of reward functions could have been used instead of a set of possible θ. However, θ is introduced since searching for the correct θ is more practical than searching for the correct reward function. The reward function is here said to be parameterized.

A MDP modified to fit in the CIRL framework is referred to as a CIRL Game. Hadfield-Menell et al. (2016) defines it in their paper and the same definition is used here.

Definition 2 (CIRL Game) *A two-agent Markov game M with identical payo s between a human or principal, $\boldsymbol{H}$, and a robot or AI, $\boldsymbol{R}$. The game is described by a tuple, $M = S, \{A^H, A^R\}, T(\cdot|\cdot,\cdot,\cdot), \{\Theta, R(\cdot,\cdot,\cdot;\cdot)\}, P_0(\cdot,\cdot), \gamma$, with the following definitions:*

- *S a set of world states: s S.*
- *A^H a set of actions for $\boldsymbol{H}$: a^H A^H.*
- *A^R a set of actions for $\boldsymbol{R}$: a^R A^R.*
- *$T(\cdot|\cdot,\cdot,\cdot)$ a conditional distribution on the next world state, given previous state and action for both agents: $T(s\,|s, a^H, a^R)$.*
- *Θ a set of possible static reward parameters, only observed by $\boldsymbol{H}$: θ Θ.*
- *$R(\cdot,\cdot,\cdot;\cdot)$ a parameterized reward function that maps world states, joint actions, and reward parameters to real numbers. R: S × A^H × A^R × Θ → R.*
- *$P_0(\cdot,\cdot)$ a distribution over the initial state, represented as tuples: $P_0(s_0, \theta$).*
- *γ a discount factor: γ $[0, 1]$.*

In the reward function, the reward parameter θ is distinguished from the other parameters, $R(s, a^{\mathbf{H}}, a^{\mathbf{R}}; \theta)$, since it is static from the start of the game while the others change. The initial state (s_0, θ), is sampled from P_0 and consists of both a world state s_0 and the active (correct) reward parameter θ Θ. To incorporate that the AI does not know the objective, only the human is able to observe θ . The bold font notation is introduced to indicate a joint action, $\boldsymbol{a}$ = ($a^{\mathbf{H}}, a^{\mathbf{R}}$), and the set of joint actions, $\boldsymbol{\mathcal{A}}$ = $A^{\mathbf{H}}$ $A^{\mathbf{R}}$. The asterisk does in general indicate correctness or optimality, and written in combination with an action or a set of actions indicate the optimal action at the given state. For example, $\boldsymbol{a}^*$ indicates an optimal joint action and $A^{\mathbf{R}*}$ indicates the set of optimal actions for $\mathbf{H}$.

2.3.2 Game procedure

The game is initialized by sampling a starting state s_0 and a reward parameter θ . Following the initialization, at each time step, both agents act. This results in a state transition and a reward, see the box bellow.

The game does not contain a stopping criterion, but is in practice stopped and the performance is evaluated at some point. If it is not desired for the agents to act simultaneously, then we can allow a wait action that is chosen every other time step such that they instead take turns acting.

Initialize

1. Sample an initial state (s_0, θ) from P_0.

For each time step t

2.1. Both agents observe the current state s_t and selects their actions $a_t^{\mathbf{H}}$ and $a_t^{\mathbf{R}}$ independently.

2.2. Both agents receive the same reward $r_t = R(s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}}; \theta)$, without $\mathbf{R}$ observing it.

2.3. The next state s_{t+1} is sampled from the transition distribution $s_{t+1} \quad P_t(s \mid s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}})$

2.3.3 Core problem

Analogously to a standard MDP, the point of interest in a CIRL game is how to chose the actions $a_t^{\mathbf{H}}$ and $a_t^{\mathbf{R}}$. Again the goal is to maximize the expected discounted sum of rewards. But, since there are two agents, there are now also two policies, $\pi^{\mathbf{H}}$ and $\pi^{\mathbf{R}}$. Together they form a policy pair that is also referred to as the joint policy, $\boldsymbol{\pi} = (\pi^{\mathbf{H}}, \pi^{\mathbf{R}})$. Now each agent can not simply consider the current state to determine how to act but also needs to consider what the other agent will do. To incorporate this, the policies can depend on all previous states, actions and the reward parameter (for $\mathbf{H}$).

$$\pi^{\mathbf{H}} : [S \times A^{\mathbf{H}} \times A^{\mathbf{R}}]^* \times \Theta \rightarrow A^{\mathbf{H}} \tag{2.2}$$

$$\pi^{\mathbf{R}} : [S \times A^{\mathbf{H}} \times A^{\mathbf{R}}]^* \rightarrow A^{\mathbf{R}} \tag{2.3}$$

The asterisk indicates in this case a sequence of any length of the set. It can be read as: to any power (less than the current time step). In our case, this means that the policy can be a function of all states and actions in all previous time steps and the reward parameter (for $\mathbf{H}$).

Another key concept is that of value $V(s)$, which i related to the expected discounted sum of rewards. "The value of a state is the expected sum of discounted rewards under the initial distribution of reward parameters and world states" (Hadfield-Menell et al., 2016). It indicates how favorable it is to occupy a state given that actions are made according to some policy. In the CIRL setting, it has the following formula.

$$V_{P_0}^{\boldsymbol{\pi}}(s) = E_\theta \quad {}_{P_0}\left[U \middle| s_0 = s, \boldsymbol{\pi}\right] = E_\theta \quad {}_{P_0}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t^{\mathbf{H}}, a_t^{\mathbf{R}}; \theta) \middle| s_0 = s, \pi^{\mathbf{H}}, \pi^{\mathbf{R}}\right] \tag{2.4}$$

The optimal joint policy is the policy pair that maximizes the value in each state if the two agents are allowed to communicate and coordinate perfectly before $\mathbf{H}$ observes θ . This is not the same as the best possible policy pair, called the solution of a CIRL Game. The reason is that if both agents knew the objective θ , and were allowed to communicate and coordinate throughout the game, then they could calculate the solution and achieve a better result. Now instead, $\mathbf{H}$ can theoretically calculate the solution since $\mathbf{H}$ observes θ (Hadfield-Menell et al., 2016). But $\mathbf{H}$ can not mediate it to $\mathbf{R}$ in other means than through its actions. Following this policy can therefore be a poor choice for $\mathbf{H}$ if $\mathbf{R}$ is following some non-compatible policy. Instead the optimal policy to the CIRL Game involves active learning, teaching and many other aspects that the solution of the CIRL Game does not have to consider. For example, the solution to the task of cleaning a house is for both agents to start cleaning right away. But, if $\mathbf{R}$ does not know that that is the task, then the optimal policy would be to first instruct $\mathbf{R}$ to clean so that they can cooperate.

2.3.4 The difficulty of finding a good policy

To follow a policy is simple, the difficult part is to come up with a good policy. Both agents need to come up with their own policies that together maximize the received reward. There are many challenges with creating policies that can handle this. The actions made by the human generally provide little information about what the AI should do. The specific task will also greatly affect the ability of the AI to learn. For example, if it is at all possible for the human to complete the task alone, or if there exists some symmetry in the game that can be exploited.

The behaviour of the human is also important to consider. What does she do when she can not complete the task alone and the AI is not providing any help, and how does this affect the AI's understanding. There can also be problems regarding whether to prefer exploring to improve the reward function approximation or whether to exploit the current approximation although it may be wrong (Armstrong & Leike, 2016).

2.3.5 Delimitations

CIRL can exist in many different variations and the variation discussed in this paper is characterized by a few different aspects. Firstly, the states and actions are discrete and finite. Secondly, the state transition distribution T is considered to be known, or approximately known, by previous exploration of the environment. Thirdly, the goal is to approximate θ as well as possible, instead of trying to maximize the reward. However, the accumulated reward is also considered in some cases. The last point is not very excluding, since it turns out that trying to find the correct θ and maximizing the reward often rely on each other.

2.4 Q-Learning and optimality

Q-learning is a model free reinforcement learning technique that will prove useful for solving CIRL problems. For any finite MDP, it will eventually find an optimal

policy. It does that by giving a score to each action in each state, that is a measure of how good that action is at a specific state. The higher the value the better the action is. In a standard MDP, a_t is a standard action, but in CIRL it is changed to a joint action. Since Q-learning is model free, it does not require a transition model and can instead use observations, but if one is available it can be used to speed up the algorithm. The update rule for Q-learning is

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha\Big(r_t + \gamma \cdot \max_a\Big\{Q(s_{t+1}, a)\Big\}\Big). \tag{2.5}$$

Where α is the learning rate, γ is the discount factor and an observation is a tuple s_t, a_t, r_t, s_{t+1} . Here s_t is the current state, a_t is the action, r_t is the obtained reward and s_{t+1} is the next state at time $t + 1$. With enough time, Q-learning will find Q which holds the expected discounted reward for taking each action. Q is also used to define what an optimal action is. A joint action is considered optimal in a state if it has a Q-value higher than or equal to all other actions in that state. All components of an optimal joint action, and only those component actions, are considered optimal. I.e. if $\boldsymbol{a}$ is optimal then so are the components $\{a^{\mathbf{H}}, a^{\mathbf{R}}\}$. If a policy always takes an optimal action then it constitutes an optimal policy.

2.5 Bayes' theorem

Bayes' theorem, or alternatively Bayes' rule, is used for updating a prior probability with new information. For example, assume that, for any given day, there is a 30 % risk of rain. When you wake up you see that the sky is gray. You can then incorporate this observation into the prior probability of rain with Bayes' rule. Maybe you then calculate that the posterior probability of rain is 70 %.

$$P(A \mid B) := \frac{P(B \mid A)P(A)}{P(B)} \tag{2.6}$$

Here $P(A \mid B)$ is the posterior probability of observing event A given that event B happened and $P(A \mid B)$ is the direct opposite. $P(A)$ and $P(B)$ are the independent probabilities for event A and B. In the example $P(A \mid B)$ would be the posterior risk of rain, i.e. 70 %. $P(A \mid B)$ represents how likely the sky is to be gray before it rains. $P(A)$ is the probability of rain, i.e. 30 % and $P(B)$ is the independent probability of a gray sky.

2.6 Softmax

The softmax function is commonly used in AI research to convert a set of real valued numbers into probabilities. Assuming a set of number, x_i R, the softmax function converts these into a new set of numbers, p_i (0, 1], that all together add up to one, $\sum_i p_i = 1$.

$$p_i := \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.7}$$

3

Analysis

θ can represent many different mathematical object and take any shape and size. The specifics change the appropriate approach to finding θ . Therefore, this chapter is divided into sections depending on what type of θ representation is being discussed. Θ is only known by $\mathbf{R}$ in the simplified version of the problem, however θ's shape and size is always known and fixed for a specific problem. In the LfD setting, demonstrations are provided by a human expert. This can be interpreted in two different ways in the CIRL setting. Either the expert is playing optimally with or without knowing what $\mathbf{R}$ will do. $\mathbf{H}$ can also behave in many other ways as will be discussed in the first section of this chapter.

3.1 Assumptions

Of major importance in CIRL is what assumptions are made about the behaviour of $\mathbf{H}$. In IRL the standard assumption is that $\mathbf{H}$, also known as the expert, is approximately following the optimal policy π . In CIRL, $\mathbf{H}$ only has control over one part of the joint action and the optimallity of action $a^{\mathbf{H}}$ depends on $a^{\mathbf{R}}$. As a result, an analogous assumption can not directly be made in CIRL. Instead the first step is to consider what $\mathbf{H}$ assumes about the policy of $\mathbf{R}$, $\pi^{\mathbf{R}}$. There are many possible such assumptions, where some are listed below. For example, $\mathbf{H}$ can assume that $\pi^{\mathbf{R}}$ $A^{\mathbf{R}}$, which is that $\mathbf{R}$ always takes an action that is part of an optimal joint action.

When $\mathbf{H}$'s assumption about $\pi^{\mathbf{R}}$ is determined, an assumption about the optimality of $\mathbf{H}$ can be made. Also here are many possibilities and some of them are listed below.

3.1.1 Overview of assumptions

For several reasons, it is infeasible to create an accurate model of how a human would behave in a cooperative task. People can have different knowledge and experience about the task. They can be differently prone to make mistakes and can in general not be expected to know how the AI behaves. As a result, the assumed behaviour of $\mathbf{H}$ is simplified and approximated. Some possible assumptions are listed here, they will later be used to create explicit models of human behaviour. The list of assumptions does not come close to being exhaustive. Instead, they are chosen to be simple and to hopefully yield interesting results.

The human's assumptions about the AI

A1.1 $\mathbf{R}$ knows what $\mathbf{H}$ will do and acts optimally accordingly, i.e. $\mathbf{R}$ solves

$$\max_{\pi^{\mathbf{R}}} E\Big[U|\pi^{\mathbf{H}}, \pi^{\mathbf{R}}, \theta\ \Big].$$

A1.2 $\mathbf{R}$ is independently behaving optimally with respect to θ , i.e. the policy is part of some optimal joint policy

$$\pi^{\mathbf{R}} \quad \operatorname*{argmax}_{\pi} E\Big[U|\boldsymbol{\pi}, \theta\ \Big].$$

A1.3 In each state, $\mathbf{R}$ will repeat the same action as done when previously visiting the state,

$$\pi^{\mathbf{R}}(s_t) = a_t^{\mathbf{R}}.$$

Where t is the time state s_t was last visited. Remember that this is only what the human is expected to assume, it does not need to be strictly followed.

A1.4 In each state, $\mathbf{R}$ will stochastically select an action based on the observed selection frequency. For example, if a_1 has been selected three times and a_2 one time, then action a_1 has a three times greater probability of being selected the next time the state is visited,

$$\pi^{\mathbf{R}}(s_t) \quad P(a^{\mathbf{R}}|s_t) = \frac{\nu(s_t, a^{\mathbf{R}})}{\sum_{a\ \ A^{\mathbf{R}}} \nu(s_t, a\ )}.$$

Where $\nu(s, a)$ is the number of times action a has been played in state s.

Assumptions about the human's optimality

A2.1 $\mathbf{H}$ selects with uniform probability one action that is the best response to the assumed action of $\mathbf{R}$,

$$\pi^{\mathbf{H}}(s_t) \quad \operatorname*{argmax}_{a^{\mathbf{H}}\ A^{\mathbf{H}}} \Big\{ Q_\theta\ (s_t, a^{\mathbf{H}}, a^{\mathbf{R}}) \Big\}.$$

A2.2 The probability of $\mathbf{H}$ to select an action is relative to how close to optimal that action is. Each actions corresponding Q-value can be used to determine this probability. An action with a higher value would be chosen with a higher probability,

$$\pi^{\mathbf{H}}(s_t) \quad P(a^{\mathbf{H}}|s_t) \quad Q_\theta\ (s_t, a^{\mathbf{H}}, a^{\mathbf{R}}).$$

A2.3 $\mathbf{H}$ chooses actions in a way as to better teach $\mathbf{R}$ about θ (this is an informal assumption, see policy `Teaching` in Appendix A for an implementation).

3.1.2 Assumption reasonableness

To create a framework for CIRL, there needs to be one assumption of each type, i.e. one about $\mathbf{H}$'s assumption about $\mathbf{R}$'s behavior and one about the optimality of $\mathbf{H}$. Since the goal is to create a realistic framework, the reasonableness of these assumptions is discussed. They are not all realistic assumptions but that does not mean that $\mathbf{R}$ can not learn from a human agent with those assumptions.

A1.1: is not realistic because $\mathbf{R}$ has for no level of human optimality any way of knowing how $\mathbf{H}$ will act beforehand.

A1.2: is not realistic since $\mathbf{R}$ does not observe θ .

A1.3 & 1.4: are reasonable if a default action or probability is assumed in the non-visited states. However, it may be hard for $\mathbf{H}$ to remember and act in accordance with this.

A2.1: is not realistic since it requires $\mathbf{H}$ to know all the optimal actions and never make a mistake.

A2.2: is not realistic but may be a good approximation depending on how exactly the probabilities are constructed.

A2.3: can be realistic in some scenarios where such actions are easy to identify.

All assumptions will be tested and compared in different settings and combinations in this report. The main focus and the basis for the developed frameworks will be on assumption A1.1 in combination with A2.2.

3.1.3 Assumed policies of H

To rigorously test the effect of these assumptions, they need to be defined mathematically as policies for $\mathbf{H}$. Note that in many cases, there are more than one optimal action. This does not affect the stochastic policies but for deterministic policies ties are broken at random (making them not fully deterministic).

Each pair of assumptions result in a different policy for $\mathbf{H}$. The algorithms in this report assume that assumptions A1.1 and A2.2 are correct. Notice that even though they are both deemed unrealistic that does not imply that they are not good assumptions to make. For example, assumption A1.1 states that $\mathbf{H}$ expects $\mathbf{R}$ to always respond optimally to the action $\mathbf{H}$ made. It is unrealistic that $\mathbf{R}$ will do that. However, it is not unrealistic for a human to make this assumption and behaving as if it were true may be efficient for teaching $\mathbf{R}$.

To create the assumed human policy, the softmax function is used as a heuristic to turn Q-values into probabilities. For a given state all possible actions are given a nonzero probability of being chosen. This probability is dependent on the Q-values of all those actions joined with their respective best action pairs for $\mathbf{R}$.

$$\pi^{\mathbf{H}} \quad P(a^{\mathbf{H}} \mid s) := \frac{\exp\left(\eta \cdot Q_\theta\ (s, a^{\mathbf{H}}, a^{\mathbf{R}})\right)}{\sum_{a^{\mathbf{H}}\ A^{\mathbf{H}}} \exp\left(\eta \cdot Q_\theta\ (s, a^{\mathbf{H}}, a^{\mathbf{R}})\right)} \tag{3.1}$$

$a^{\mathbf{R}}$ is shorthand notation for saying that this action is the optimal response to $a^{\mathbf{H}}$, i.e. it is the optimal action for $\mathbf{R}$ given $a^{\mathbf{H}}$. $\eta \quad [0, \infty)$ is an optimality parameter. With a high value, the probability for $\mathbf{H}$ to chose a good action is also high. It is used to change how close to optimal $\mathbf{H}$ behaves. Q_θ holds the precise expected sum of discounted rewards for an optimal policy and the correct reward parameter. Q-values are used since they are well established and easy to use. This policy is referred to as `ApproxOpt`. The other tested policies are called `Optimal`, `Teaching`, `IndeOpt`, `Repeating` and `FreqRep`. They are based on some of the other assumptions listed above. When acting in accordance with `Optimal`, $\mathbf{H}$ chooses a random optimal action. `Teaching` is similar to `Optimal`, but actions are chosen in a way as to provide more information. `IndeOpt` is another kind of optimal behaviour that considers $\mathbf{R}$'s available actions in grater detail. In `Repeating` and `FreqRep`, the choice of action are affected by the previous choice in the same state. The definitions of all policies and what assumptions they use can be found in Appendix A.

3.2 θ as a finite index

The first scenario to be analyzed is the simplified case when θ is a finite index, $|\Theta| = n$ with $0 < n < \infty$. In this case $\mathbf{R}$ is also allowed to observe Θ. Since the transition distribution is known, the optimal Q can be calculated for each θ before training starts. The notation Q_θ is introduced for this purpose. It holds the precise expected sum of discounted rewards for an optimal policy and the reward parameter θ. The idea is to look at what actions $\mathbf{R}$ could have taken and what actions $\mathbf{H}$ took to compare that to what $\mathbf{H}$ probably would have taken when pursuing each possible θ. All in line with the `ApproxOpt` policy. This can be used to gradually update the belief about which θ is correct. There also needs to be a way of deciding what action $\mathbf{R}$ should take to learn the most about θ and accumulate reward.

In this scenario, a probability distribution over the possible θ is introduced. This distribution is denoted $\phi_t^{(\theta)}$. Since θ is sampled from the P_0 distribution, $\phi_0^{(\theta)}$ should be initialized to the same distribution

$$\phi_0^{(\theta)} = P_0(\theta), \quad \theta \quad \Theta. \tag{3.2}$$

$\phi_t^{(\theta)}$ is then updated in each time step.

These ideas are translated into the code used in the experiments. Directions of where to find the code can be found in Appendix B.

3.2.1 Testing environment

CIRL is a relatively new idea and no standard benchmark exists for testing algorithms. The game used by Palaniappan et al. (2017) could technically be used, but since the setting of the algorithms are different the results can not be compared directly. Therefore, a new game is created specifically for this setting.

The introduced testing environment is a simple deterministic grid game referred to as `SimpleGridGame`, see Figure 3.1. The game consists of a grid with 5 × 3 squares. The agents each occupy one square (green) and can move between them. The agents available actions are $A^{\mathbf{H}} = \{\leftarrow, \rightarrow, wait\}$ and $A^{\mathbf{R}} = \{\leftarrow, \rightarrow, \uparrow, \downarrow, wait\}$. The arrows indicate a step made in the corresponding direction and the $wait$ action makes the agent stay where it is. Note that although the same notation has been used for simplicity on some actions for both $\mathbf{H}$ and $\mathbf{R}$, they are not technically the same. $\mathbf{H}$ is limited to moving along the top most row while $\mathbf{R}$ can move around in the bottom two rows. The game has 4 possible reward parameters, $|\Theta| = 4$. Each with a reward of 1 when both agents are occupying the corresponding squares (yellow). Each reward parameter has an equal chance of being selected and at the start of the game one of them is chosen at random. The choice is observed by $\mathbf{H}$ but not by $\mathbf{R}$. For example, if $\theta_2 = \theta$ then the goal is for both players to move to their corresponding square labeled with θ_2. If they both occupy them at the same time then they receive a reward. They need to cooperate since only $\mathbf{H}$ knows where $\mathbf{R}$ should move and neither of them will gain anything from $\mathbf{H}$ moving to the goal alone. The dilemma, in this game, is that for all reward parameters the goal square is the same for $\mathbf{H}$. This is a deliberate design choice since it will test $\mathbf{R}$'s ability to draw conclusions from minor differences in $\mathbf{H}$'s actions that depend on θ .

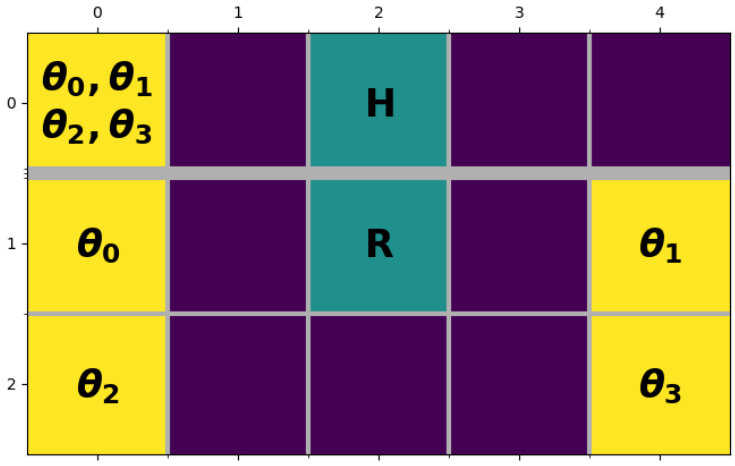


Figure 3.1: The testing environment for the simplified setting. $\mathbf{H}$ and $\mathbf{R}$ can take a step in any direction at each time step but may not cross the wide line separating row 0 and 1. The shown configuration is the starting state s_0. The goal is for both agents to move to the correct yellow squares.

3.2.2 Updating beliefs

The assumption is made that the probability of $\mathbf{H}$ selecting an action is proportional to the action's Q-value compared to the other available actions' Q-values, as specified by policy `ApproxOpt`. These probabilities can be used to update $\mathbf{R}$'s beliefs over θ in a Bayesian fashion. A minor modification to the `ApproxOpt` policy results in a likelihood update for each θ

$$P_\theta(a_t^{\mathbf{H}} \mid s_t) := \frac{\exp\left(\eta \cdot Q_\theta(s_t, a_t^{\mathbf{H}}, a^{\mathbf{R}})\right)}{\sum_{a^{\mathbf{H}} \; A^{\mathbf{H}}} \exp\left(\eta \cdot Q_\theta(s_t, a^{\mathbf{H}}, a^{\mathbf{R}})\right)}. \tag{3.3}$$

$P_\theta(a_t^{\mathbf{H}} \mid s_t)$ is the likelihood that $\mathbf{H}$ would have chosen action $a_t^{\mathbf{H}}$ given state s_t and that the reward parameter is the correct one, $\theta = \theta$. $a^{\mathbf{R}}$ is the optimal action for $\mathbf{R}$ given $a^{\mathbf{H}}$. Finally, $\eta \geq 0$ is the optimality parameter that determines how much weight is put on each observation. This likelihood can now be used together with Bayes rule to update ϕ.

$$\phi_{t+1}^{(\theta)} := \frac{\phi_t^{(\theta)} P_\theta(a_t^{\mathbf{H}} \mid s_t)}{\sum_{\theta \;\; \Theta} \phi_t^{(\theta)} P_\theta \, (a_t^{\mathbf{H}} \mid s_t)} \tag{3.4}$$

For policies `Repeating` and `FreqRep`, a slightly di□erent likelihood is used that takes into account the assumption of repetition. These likelihoods can be found in Appendix A.

3.2.3 Action selection

The objective of $\mathbf{R}$ is to maximize the reward, and the highest Q-value informs $\mathbf{R}$ about what action to take. But, a problem is that the Q-values are di□erent for

each θ, and θ is only estimated with a probability distribution. It is therefore not clear what action to select. Instead the action with the highest expected value can be chosen.

$$\begin{aligned}\pi^R(s_t) :&= \underset{a^{\mathbf{R}} \in A^{\mathbf{R}}}{\operatorname{argmax}} \Big\{ \mathrm{E}\big[Q\big] \Big\} \\ &= \underset{a^{\mathbf{R}} \in A^{\mathbf{R}}}{\operatorname{argmax}} \Big\{ \mathrm{E}\big[\sum_{\theta \in \Theta} Q_\theta(s_t, \pi^{\mathbf{H}}, a^{\mathbf{R}}) \phi_t^{(\theta)} \big] \Big\} \\ &= \underset{a^{\mathbf{R}} \in A^{\mathbf{R}}}{\operatorname{argmax}} \Big\{ \sum_{\theta \in \Theta} \sum_{a^{\mathbf{H}} \in A^{\mathbf{H}}} Q_\theta(s_t, a^{\mathbf{H}}, a^{\mathbf{R}}) P_\theta(a^{\mathbf{H}} \mid s_t) \phi_t^{(\theta)} \Big\}\end{aligned} \tag{3.5}$$

Similarly to the likelihood, $P_\theta(a^{\mathbf{H}} \mid s_t)$ is the probability for $\mathbf{H}$ to choose action $a^{\mathbf{H}}$ given that $\theta = \theta$. $\pi^{\mathbf{H}}$ is referred to as a non-static policy since the action in a given state may change when $\phi^{(\theta)}$ changes.

A problem is that this is a purely exploitative approach to selecting an action, which is fine if $\mathbf{H}$'s actions clearly indicate the correct θ. But if they do not provide any guidance, then $\mathbf{R}$ needs to do something else. Since observations in new states can provide new information, $\mathbf{R}$ should explore those states.

3.2.4 Promoting exploration

In some situations, $\mathbf{R}$ will actively have to act in a way to learn more from $\mathbf{H}$. This can for example happen if the action with the highest expected Q-value is to not do anything ($wait$) and the actions $\mathbf{H}$ makes does not provide any new information. Then exploration can the be useful. It can be carried out in many ways and at different times. ϵ-greedy is one well known method of exploration. This method chooses a random action in each time step with probability ϵ. However, as Θ is known, there are likely more efficient approaches to exploration.

3.2.4.1 When to explore

Exploration is less useful when $\mathbf{R}$ is certain about θ , and more useful when $\mathbf{R}$ is uncertain. It is also not necessary when the actions $\mathbf{H}$ take provide enough information on their own. Comparing $\phi_{t-1}^{(\theta)}$ and $\phi_t^{(\theta)}$ indicates whether the belief for any θ has been increased. If the probability of the most likely θ is constantly increasing by at least some constant factor, then $\mathbf{R}$ is learning more. If not then it may be better to explore.

Two exploration parameters are introduced here together with the respective values used in this report. The values are specifically chosen for `SimpleGridGame` and selected based on experience to yield good results. κ = 50% is the threshold for exploration. If any θ has a probability that is larger then κ of being correct no exploration will be done. δ = 2% is the minimum increase in probability for exploration not to be preferred. I.e. if the most likely θ is not at least δ more likely after a time step then $\mathbf{R}$ should explore.

3.2.4.2 How to explore

If $\mathbf{R}$ pursues the correct reward parameter θ , then $\mathbf{H}$ should start performing approximately optimal with regards to that θ. If $\mathbf{H}$ does not, then it is likely not the correct reward parameter. This is an insight for more e□cient exploration. If it is indicated that exploration is needed by the exploration parameters, then $\mathbf{R}$ could randomly sample a θ based on ϕ. $\mathbf{R}$ can then behave optimally with respect to the randomly chosen θ and see how $\mathbf{H}$ reacts, this method is called Thompson sampling (Russo, Roy, Kazerouni, & Osband, 2017). It is not simple to determine for how long to pursue θ. Therefore, a method of duration doubling can be used. It doubles the exploration duration each time exploration is needed.

The exploration policy is

$$\pi^R(s_t) := \underset{a^{\mathbf{R}} \; A^{\mathbf{R}}}{\operatorname{argmax}} \left\{ Q_\theta \left(s_t, a^{\mathbf{H}}, a^{\mathbf{R}}\right) \right\} \tag{3.6}$$

where θ is the randomly sampled reward parameter and $a^{\mathbf{H}}$ is an action that is optimal in pair with $a^{\mathbf{R}}$.

3.3 θ as a matrix over states

Limiting θ to a finite set can be simple and practical for some applications. It does however also limit the application areas. All rewards are functions of the state and joint action. The space of all possible reward functions can therefore be covered by a matrix representing the reward for all state and joint action pairs, $\theta(s, a^{\mathbf{H}}, a^{\mathbf{R}})$ R. But, due to a lack of time, the problem will be simplified to only cover a general reward function over the state, $\theta(s)$. It should be possible to expand the methodology to cover the truly general case without too much e□ort.

When only the shape of θ is known there is an infinite number of possible reward parameters and Q_θ can therefore not be calculated for all of them. What can be done instead is to use Q-learning to get a form of measurement of how e□ective an action is at changing the state closer to every other state. This can be used to guess where $\mathbf{H}$ is moving towards. For every observation, $\mathbf{R}$ can then update the estimation $\theta_t(s)$. All policies will be tested in this setting as well.

3.3.1 Testing environment

The introduced testing environment for the general problem is also a deterministic grid game. It is referred to as `GeneralGridGame` and can be seen in Figure 3.2. The game is designed to allow for infinitely many reward functions and to be constantly changing, so that the agents are forced to move around. It consists of a grid with 6 × 3 squares. The agents each occupy one square (green and blue) and they can move in the grid. The agents available actions are $A^{\mathbf{H}} = A^{\mathbf{R}} = \{\leftarrow, \rightarrow, wait\}$. The arrows indicate a step made in the corresponding direction and the $wait$ action makes the agent stay where it is. Note that, although the same notation has been used for simplicity on some actions for both $\mathbf{H}$ and $\mathbf{R}$, they are not technically the same. $\mathbf{H}$ is limited to moving along the topmost row and $\mathbf{R}$ in the middle row.

There is also a yellow square moving back and forth along the bottom row. Figure 3.2 shows the starting state. The yellow square moves all the way to the right one step at a time, then waits one turn and moves all the way back. When it is back it waits one turn and then repeats the process indefinitely.

There is now no finite set of possible reward parameters. Instead the reward can be any function of the state. The goal for $\mathbf{R}$ is therefore to find the correct reward for each state, i.e. find the values θ (s), s S. However, $\mathbf{H}$ needs to have a reward parameter in mind to teach it to $\mathbf{R}$. Therefore, a reward parameter is chosen and used for all experiments. It is the one with a reward of 1.0 when both agents and the yellow square all occupy the same column and 0.0 otherwise. The agents need to cooperate since they both have to be in the same column as the yellow square at the same time for them to receive a reward. This information has to be mediate to $\mathbf{R}$ by $\mathbf{H}$.

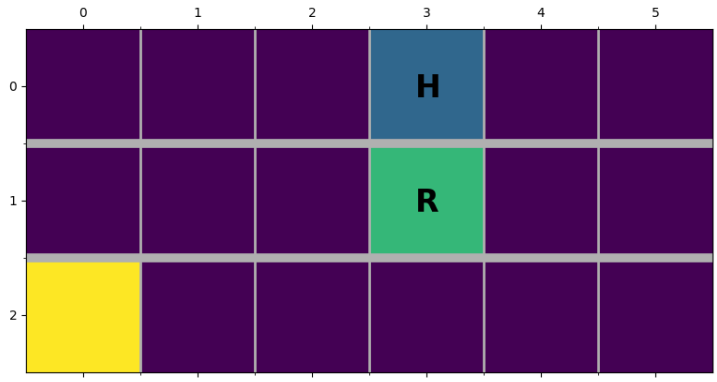


Figure 3.2: The testing environment for the general setting. $\mathbf{H}$ and $\mathbf{R}$ can move left, right or wait at each time step. They are both limited to moving in their respective rows. The shown configuration is the starting state s_0 S. The goal is for both agents to occupy the same column as the yellow square. This is complicated by the fact that it keeps moving back and forth.

3.3.2 Action efficiency measure

$\mathbf{R}$ can not measure how much aligned the actions of $\mathbf{H}$ are with the di□erent θ since there are infinitely many possible θ. What $\mathbf{R}$ can measure is how e□ective the actions taken by $\mathbf{H}$ are at changing the state closer to any other states. It is a measurement that relates all states to each other by how fast the state can change from one to the other. The time is indicated for each action in every state. It is an abstract measurement and the time is calculated by taking a specific action in a state and thereafter following the optimal path. Returning to the example of folding laundry, this measurement would indicate how e□ective an action like placing the table cloth on the ironing board is for making the first fold (note very), vice versa or any other combination of states and actions.

The measurement is represented by q ($s, a^{\mathbf{H}}, a^{\mathbf{R}}, s$) where ($a^{\mathbf{H}}, a^{\mathbf{R}}$) is the joint action made in state s and s is the goal state. The values of q are produced by

performing Q-learning a total of |S| times, one for each possible goal state. Each time the reward for moving to state s is 1.0, s is also an absorbing state.

$$q\ (s, a^{\mathbf{H}}, a^{\mathbf{R}}, s\ ) := Q_s\ (s, a^{\mathbf{H}}, a^{\mathbf{R}}) \tag{3.7}$$

Where Q_s is the result of Q-learning with the reward

$$R(s, a^{\mathbf{H}}, a^{\mathbf{R}}; \theta) = \begin{cases} 1 & s = s \\ 0 & \text{otherwise} \end{cases}. \tag{3.8}$$

The result is that q holds values in range [0, 1] representing the effectiveness of taking actions $(a^{\mathbf{H}}, a^{\mathbf{R}})$ in state s for reaching state s. The discount γ has to be strictly less than one for this to work, otherwise all actions are equally optimal.

3.3.3 Updating beliefs

$\mathbf{R}$ updates the approximation $\theta_t(s)$ with every observation. To simplify the problem the approximated rewards are, without loss of generality, scaled to the interval between minus one and one, $\theta_t(s) \quad [-1, 1]$. This can be done since the rewards of θ are finite. $\theta_t(s)$ may then not be similar to $\theta\ (s)$, but the policy derived from it hopefully will.

$\mathbf{H}$ is again assumed to follow the `ApproxOpt` policy. However, due to lack of time an ideal way to update $\theta_t(s)$ is not searched for. Instead, $\theta_t(s)$ is updated with a simple heuristic that for each s compares the most and the least effective action, for moving to s, with the action chosen by $\mathbf{H}$. Unless all actions are equally effective, this creates an interval $[b^{low}, b^{high}]$ that is mapped to $[-1, 1]$. The action performed by $\mathbf{H}$ is then mapped to a value in this interval. Over time all values from all observations are averaged. The idea is that states that $\mathbf{H}$ mostly moves towards get a value close to 1.0. For states that $\mathbf{H}$ is indifferent about the value averages close to 0.0. States that $\mathbf{H}$ avoids get a value close to -1.0.

$$\begin{aligned} b_t^{low}(s\ ) &= \underset{a^{\mathbf{H}}}{\text{minimum}}\ \underset{A^{\mathbf{H}},\, a^{\mathbf{R}}}{\text{minimum}}\ \underset{A^{\mathbf{R}}}{\text{minimum}} \left\{ q\ (s_t, a^{\mathbf{H}}, a^{\mathbf{R}}, s\ ) \right\} \\ b_t^{high}(s\ ) &= \underset{a^{\mathbf{H}}}{\text{maximum}}\ \underset{A^{\mathbf{H}},\, a^{\mathbf{R}}}{\text{maximum}}\ \underset{A^{\mathbf{R}}}{\text{maximum}} \left\{ q\ (s_t, a^{\mathbf{H}}, a^{\mathbf{R}}, s\ ) \right\} \\ w_t(s\ ) &= \underset{a^{\mathbf{R}}}{\text{maximum}}\ \underset{A^{\mathbf{R}}}{\text{maximum}} \left\{ q\ (s_t, a_t^{\mathbf{H}}, a^{\mathbf{R}}, s\ ) \right\} \end{aligned} \tag{3.9}$$

$b_t^{low}(s\ )$ is the effectiveness of the least effective action for changing the state closer to s, whereas $b_t^{high}(s\ )$ is the most effective. $w_t(s\ )$ is the value of the action that $\mathbf{H}$ chose, in pair with the best action $\mathbf{R}$ could have chosen. If $b_t^{low}(s\ ) = b_t^{high}(s\ )$, then $\mathbf{H}$ had to make a choice between actions and $\theta_t(s\ )$ can be updated according to

$$\begin{aligned} \alpha_s &= \frac{\nu(s\ )}{\nu(s\ ) + 1} \\ \theta_t(s\ ) &= \alpha_s\, \theta_{t-1}(s\ ) + (1 - \alpha_s) \left(2 \cdot \frac{w_t(s\ ) - b_t^{low}(s\ )}{b_t^{high}(s\ ) - b_t^{low}(s\ )} - 1 \right) \end{aligned} \tag{3.10}$$

Here $\nu(s)$ is the number of times $\theta_t(s)$ has been updated. α_s is the learning rate equal to taking the average of all updates. The bottom formula is the mapping from $[b^{low}, b^{high}]$ to $[-1, 1]$ and update. The result is that every state has an approximated reward attributed to it that is the average of the values w_t $[-1, 1]$. For policies `Repeating` and `FreqRep` a slightly different heuristic is used that takes into account the assumption of repetition. These heuristics can be found in Appendix A.

3.3.4 Action selection

Since $\mathbf{R}$ always has an approximation of the reward function, $\theta_t(s)$, Q-learning can be used to select actions. The approximation is updated in each time step and therefore $Q_t(s, a^{\mathbf{H}}, a^{\mathbf{R}})$ also has to be updated in each time step to utilize the latest approximation. It is done by running the Q-learning algorithm again and $Q_t(s, a^{\mathbf{H}}, a^{\mathbf{R}})$ is now indexed by the time step to highlight that it changes. Usually, this takes time but since $\theta_t(s)$ only changes a little in each update, Q-learning can use $\theta_{t-1}(s)$ for initialization, making it complete much faster.

Taking the action with the highest Q-value results in the policy

$$\pi^R(s_t) := \underset{a^{\mathbf{R}} \ A^{\mathbf{R}}}{\operatorname{argmax}} \left\{ Q_t(s_t, a^{\mathbf{H}}, a^{\mathbf{R}}) \right\} \tag{3.11}$$

where all ties are broken randomly and $a^{\mathbf{H}}$ is an action that is optimal in pair with $a^{\mathbf{R}}$. This policy greedily tries to maximize the reward, which is good if $\theta_t(s)$ is an accurate enough approximation so that $\pi^{\mathbf{R}}$ is an optimal policy. It is also an acceptable policy if the actions made by $\mathbf{H}$ provide enough information for $\theta_t(s)$ to eventually be an accurate enough approximation. But if it does not provide enough information, then $\mathbf{R}$ needs to explore.

The method called ϵ-greedy, also mentioned in Section 3.2.4, is used to introduce an exploration behaviour. This method of exploration is chosen because of its simplicity. During the experiments on `GeneralGridGame`, a value of 0.1 is used for ϵ since it is a commonly used value (Sutton et al., 1998).

4

Experiments

This chapter explains the experimental setup, the different performance measurements and the actual results. The setup is different for the two frameworks and the chapter is therefore divided along that line. Important observations are highlighted and some effort is put into decoding what $\mathbf{R}$ learns.

4.1 θ as a finite index

The experiments are designed to test if $\mathbf{R}$ can reliably find θ and, if successful, how long it takes. $\mathbf{R}$ has a probability distribution over the different θ and can never become completely certain about which is correct, since $\mathbf{H}$ is assumed to make mistakes. It is $\mathbf{R}$'s assigned probability of θ being correct, $\phi^{(\theta)}$, that is measured, together with the median time to get the first reward. The median is chosen since in some cases a reward is never received. All experiments regarding a finite theta are carried out in `SimpleGridGame`.

4.1.1 Illustrative example

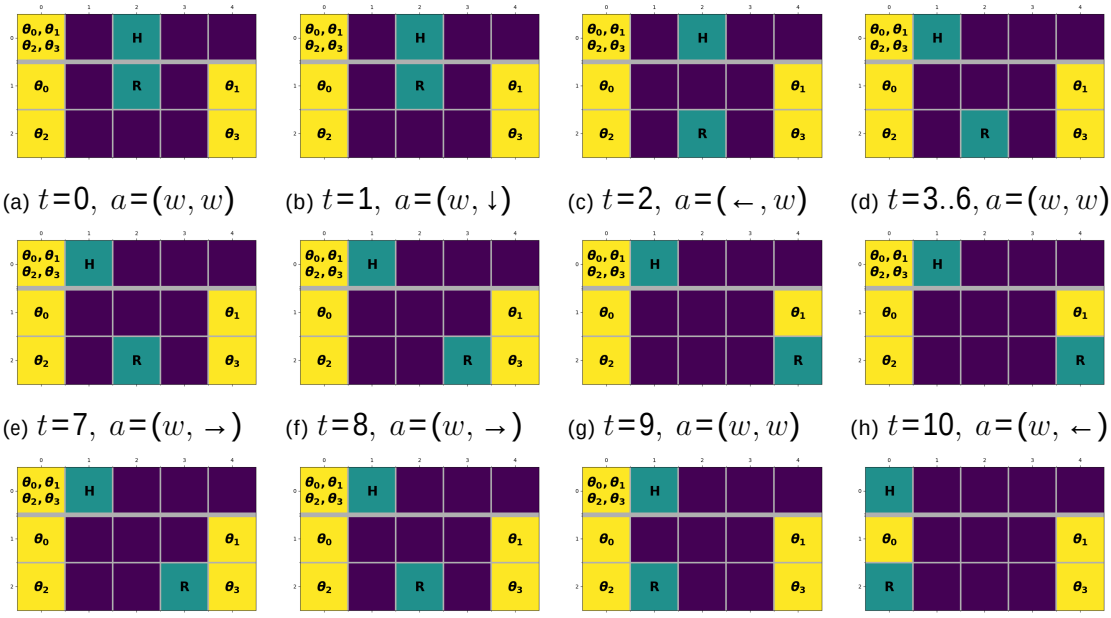
An example round of `SimpleGridGame` is used to illustrate the underlying behavior that causes the results in the next section. It is a short round of the game, ended the moment any positive reward is received. $\mathbf{H}$ is following the `Teaching` policy and the result is that $\mathbf{R}$ assigns the highest probability to the correct θ, see Figure 4.1 and Table 4.1.

4.1.2 Comparison of assumptions

What is sought after is learning when $\mathbf{H}$ follows the `ApproxOpt` policy. However, since the behaviour of $\mathbf{H}$ is integral to learning, the effect on the performance of all discussed polices will be measured. The experiments are run 10 000 times since the confidence, $\phi^{(\theta)}$, goes up and down with the mistakes that $\mathbf{H}$ makes and the variance is quite large. All simulations lasts for 60 time steps to guarantee that all interesting learning behaviour is captured. Unlike in the illustrative example a high reward can here be accumulated.

4.1.2.1 Testing incorrect optimality assumptions

The parameter η can be seen as either the assumed optimality of $\mathbf{H}$, or the inference strength of each observation for $\mathbf{R}$. The parameter does not have a clear connection



(a) t=0, a=(w, w) (b) t=1, a=$(w, \downarrow)$ (c) t=2, a=$(\leftarrow, w)$ (d) t=3..6, a=(w, w)

(e) t=7, a=$(w, \rightarrow)$ (f) t=8, a=$(w, \rightarrow)$ (g) t=9, a=(w, w) (h) t=10, a=$(w, \leftarrow)$

(i) t=11, a=$(w, \leftarrow)$ (j) t=12, a=$(w, \leftarrow)$ (k) t=13, a=$(\leftarrow, \leftarrow)$ (l) t=14

Figure 4.1: A typical round of `SimpleGridGame` with the `Teaching` policy. Each snapshot is captioned with the corresponding time step and the actions taken by the agents, $a = (a^{\mathbf{H}}, a^{\mathbf{R}})$. The correct reward parameter is θ_2. Following the `Teaching` policy makes $\mathbf{H}$ not move closer to the goal unless $\mathbf{R}$ is equally far from it. Therefore, $\mathbf{H}$ chooses the $wait$ action in the first two time steps. $\mathbf{H}$ could be even more clear by moving away from the goal when $\mathbf{R}$ moves towards an incorrect goal. During time steps 3,..,6, no agent makes any move. This causes $\mathbf{R}$ to become more confident that one of the goals on the bottom row is correct. At time step 7, $\mathbf{R}$ chooses to explore θ_3. This does not immediately provide any new information. Therefore, $\mathbf{R}$ explores again in the next time step. θ_3 is randomly chosen again and now explored for two time steps. This is a shortened game that ends when the first reward is received.

to the real world and will have to be approximated. The result of using an incorrect value, i.e. $\eta^{\mathbf{R}} = \eta^{\mathbf{H}}$, is therefore explored here. Figure 4.2 shows the result when $\mathbf{R}$ assumes a value, lower or higher, than the correct value. Table 4.2 shows the median time it takes until the first reward is received. They quite clearly shows that when $\mathbf{R}$ assumes a too high value, the learning speed increases but so does the variability. The reason is that $\mathbf{R}$ draws a conclusion directly that is most often correct but quite often wrong too. The opposite e□ect can be seen when $\mathbf{R}$ assumes a too low value. But even here the di□erence between runs is quite large which is problematic. The net e□ect is that the median time to receive the first reward is almost constant.

An incorrect conclusion is usually made when $\mathbf{H}$ makes a mistake in the first time step. $\mathbf{R}$ then presumes that the correct θ is in the other row, i.e. row 1 as opposed to 2 or vice versa. With $\eta^{\mathbf{H}} = 1.0$ and $\gamma = 0.9$ this happens quite often. It may then take a long time, if ever, before any of the θ on the other row is explored, especially if $\eta^{\mathbf{R}}$ is large. The graphs are a bit misleading since the value is very varying for a single run. Going up and down in unison with the mistakes that $\mathbf{H}$ makes. It is worth noticing the percentage of runs where $\mathbf{R}$ fail to ever attribute θ a likelihood of

t	$\phi_t^{(\theta_0)}$	$\phi_t^{(\theta_1)}$	$\phi_t^{(\theta_2)}$	$\phi_t^{(\theta_3)}$	$a_t^{\mathbf{H}}$	$a_t^{\mathbf{R}}$	Exploring
0	0.250	0.250	0.250	0.025	wait	wait	-
1	0.222	0.222	0.278	0.278	wait	↓	-
2	0.195	0.195	0.305	0.305	←	wait	-
3	0.154	0.154	0.346	0.346	wait	wait	-
4	0.119	0.119	0.381	0.381	wait	wait	-
5	0.090	0.090	0.410	0.410	wait	wait	-
6	0.067	0.067	0.433	0.433	wait	wait	-
7	0.049	0.049	0.451	0.451	wait	→	θ_3
8	0.035	0.035	0.465	0.465	wait	→	θ_3
9	0.032	0.034	0.581	0.353	wait	wait	θ_3
10	0.029	0.025	0.712	0.234	wait	←	-
11	0.025	0.018	0.813	0.144	wait	←	-
12	0.020	0.015	0.872	0.094	wait	←	-
13	0.014	0.010	0.881	0.095	←	←	-
14	0.010	0.007	0.899	0.085	-	-	-

Table 4.1: The beliefs $\mathbf{R}$ has over the different θ, and the actions both agents choose in each time step. $\phi_t^{(\theta_0)}$ and $\phi_t^{(\theta_1)}$ falls quickly towards zero but $\phi_t^{(\theta_2)}$ and $\phi_t^{(\theta_3)}$ moves asymptotically towards 50 %. At time step 7, $\mathbf{R}$ starts exploring, providing new information at time step 9. The key piece of information is that $\mathbf{H}$ does not move closer to the goal when $\mathbf{R}$ is close to θ_3, indicating that $\theta_3 = \theta$. This eliminates θ_3 as a possible assumption and left is only θ_2.

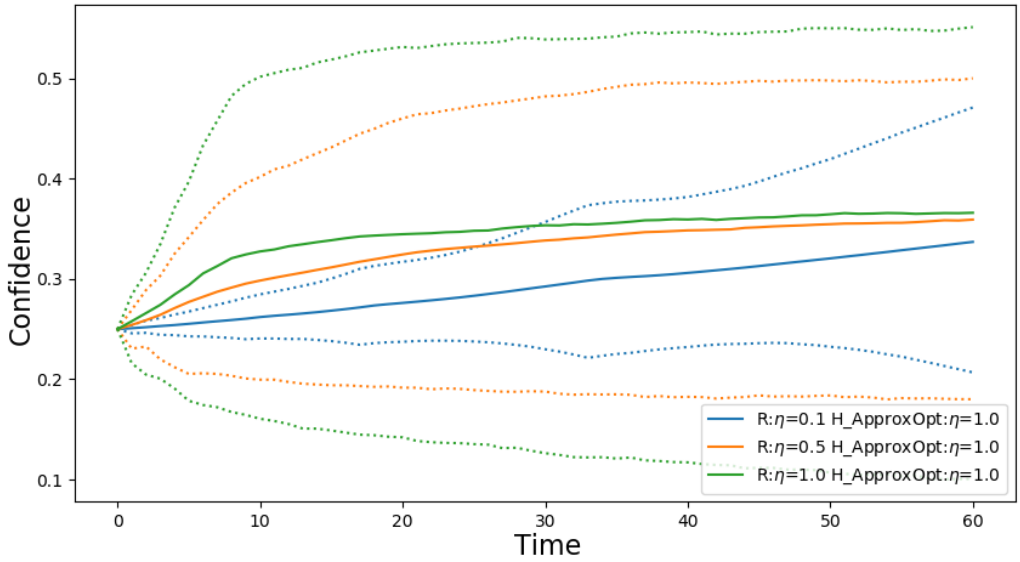
over 50 %. At $\eta^{\mathbf{H}} = \eta^{\mathbf{R}} = 1.0$ about 27 % fail to do so and at $\eta^{\mathbf{H}} = 2.0, \eta^{\mathbf{R}} = 1.0$ this value is around 18%. When η approaches infinity, `ApproxOpt` approaches `Optimal` and the results for that is presented in the next section.

$\eta^{\mathbf{R}}$:	0.1	0.5	1.0	2.0	10.0	20.0
Median time:	17	16	16	16	18	17

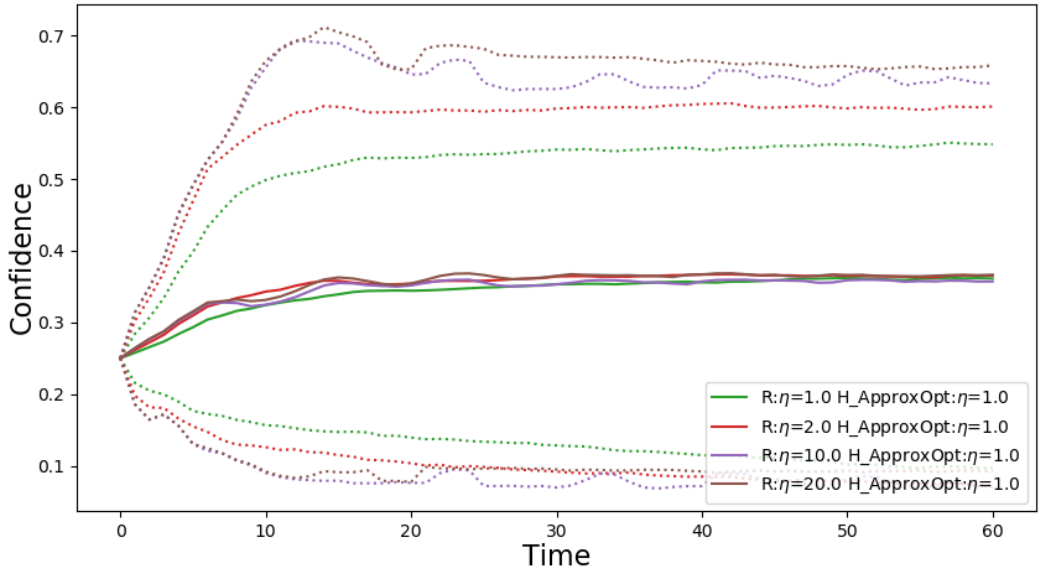
Table 4.2: The median time until the first real positive reward is received. The median is used since in some runs no positive reward is ever received. The values does not differ by any significant amount. Interestingly, there is no clear relation between the time and the value of η. But values close to the actual value of 1.0 could be better, unless it is just a fluctuation. The median is taken over 10000 runs.

4.1.2.2 Various kinds of optimal human behaviour

Three kinds of optimal behavior are proposed for $\mathbf{H}$. Their effect on the learning ability of $\mathbf{R}$ can be seen in Figure 4.3. The median time until the first reward is received is shown in Table 4.3. The `Teaching` and `Optimal` policies perform better than any other policy. `Teaching` performs the best. They both learn faster on average and does that without increasing the variability. It is also here worth noticing the percentage of runs where $\mathbf{R}$ fail to ever attribute θ a likelihood of over



(a) $\mathbf{R}$ assumes a too low value.



(b) $\mathbf{R}$ assumes a too high value.

Figure 4.2: The solid lines show the confidence $\phi^{(\theta)}$, i.e. the estimate $\mathbf{R}$ has of θ being correct. This is measured with six different values of η for $\mathbf{R}$ while the value stays the same for $\mathbf{H}$, i.e. the human agent has the same policy each time. The values are 0.1, 0.5, 1.0, 2.0, 10.0 and 20.0 for $\mathbf{R}$ and 1.0 for $\mathbf{H}$. The dotted lines show the average deviations between runs on both sides of the mean, i.e. the average result of all values lower or higher than the total average. All values are averaged over 10 000 runs of 60 time steps. Notice that the learning speed increases when $\mathbf{R}$ assumes a higher η value, but at the same time the average deviation also increases, even though it is always quite large.

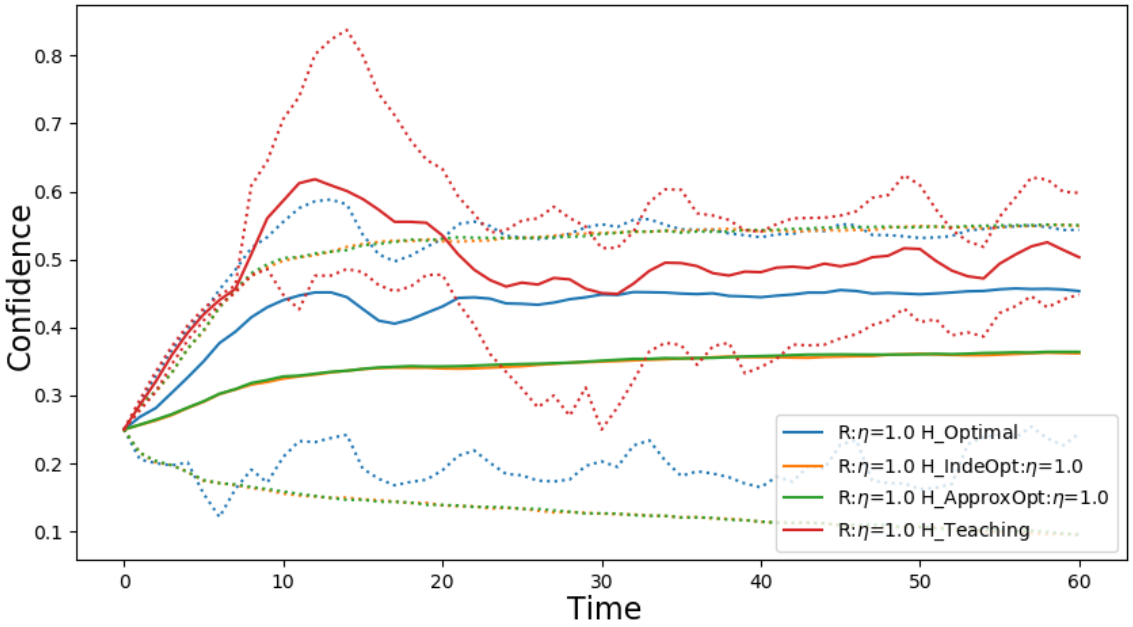


Figure 4.3: The solid lines show $\phi^{(\theta)}$, i.e. the estimate $\mathbf{R}$ has of θ being correct. This is measured for the three different optimal policies and the default `ApproxOpt` policy. $\mathbf{R}$ assumes that $\mathbf{H}$ is following `ApproxOpt` with $\eta = 1.0$. The dotted lines show the average deviations between runs on both sides of the mean, i.e. the average result of all values lower or higher than the total average. All values are averaged over 10 000 runs of 60 time steps. `IndeOpt` perform almost identically to the `ApproxOpt` policy and their lines overlap. `Optimal` performs better and `Teaching` better again, this time without increased variability. Notice also that `Teaching` achieves a very high value at first but falls down to a lower level afterwards.

50 %. For `Optimal` it is 13 % of the times. For `IndeOpt` 27 % and for `Teaching` 0 % compared to 27 % for `ApproxOpt`. Here all $\eta = 1.0$.

A new consequence of the exact formulation of the likelihood function, the use of Q-values and the game design is apparent in Figure 4.3. The estimate gets very high at first for the `Teaching` policy but later falls back down to a lower but still high level. The reason is that when both agents are at θ , $\mathbf{H}$ will stay in place. However, staying in place is equally optimal for all θ and this will cause the probabilities to even out a bit.

Policy:	`Optimal`	`IndeOpt`	`ApproxOpt`	`Teaching`
Median time:	13	16	16	12

Table 4.3: The median time until the first real positive reward is received. The median is used since in some runs, no positive reward is ever received. The policies `Optimal` and `Teaching` perform significantly better than the other. `Teaching` performs marginally better than `Optimal`. For `IndeOpt` and `ApproxOpt` it takes the same time to receive a reward. The median is taken over 10 000 runs.

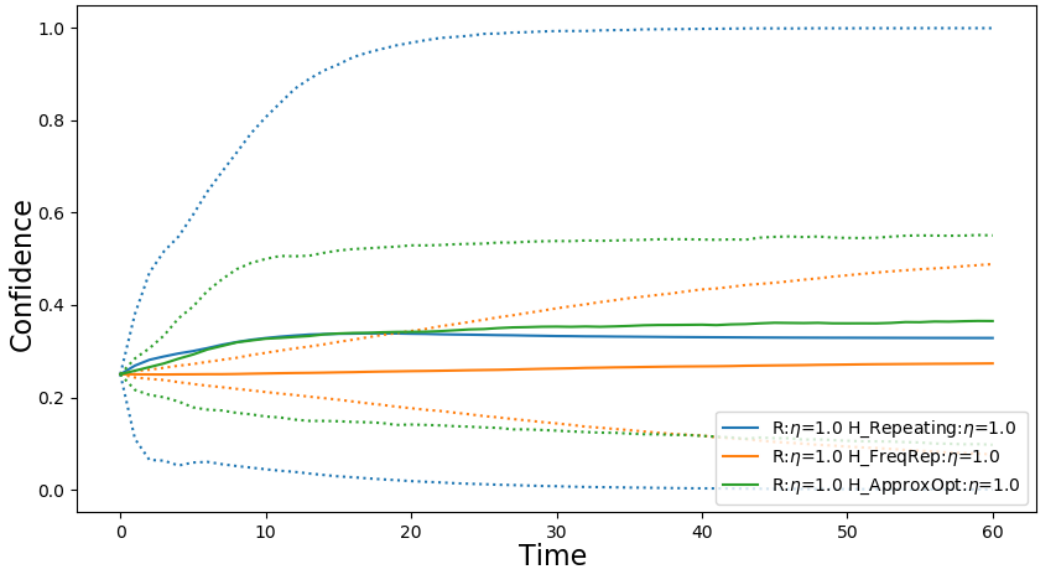


Figure 4.4: The solid lines show $\phi^{(\theta)}$, i.e. the estimate $\mathbf{R}$ has of θ being correct. It is measured for the two different policies with repeating behaviour, `Repeating` and `FreqRep`, and the default `ApproxOpt` policy. $\mathbf{R}$ is aware of the repeating behaviour and does therefore use a modified likelihood, see Appendix A. The dotted lines show the average deviations between runs on both sides of the mean, i.e. the average result of all values lower or higher than the total average. All values are averaged over 10000 runs of 60 time steps. `Repeating` has a very high variability and worse performance than `ApproxOpt` in the long run, but it does initially learn faster. `FreqRep` learns very slowly.

4.1.2.3 Repeating policies

There are two proposed policies for $\mathbf{H}$ that assumes some kind of repetition, these are `Repeating` and `FreqRep`. If $\mathbf{R}$ is aware of the repeating behaviour it could improve learning. Figure 4.4 and Table 4.4 show the results from the experiments, which quite clearly shows that neither of these polices perform well. In fact, they perform significantly worse. Less than half ever receive any reward. Interestingly though, `Repeating` initially learns faster than `ApproxOpt` but then falls down to a lower level.

Policy:	`Repeating`	`FreqRep`	`ApproxOpt`
Median time:	> 60	> 60	16

Table 4.4: The median time until the first real positive reward is received. The median is used since in some runs no positive reward is ever received. The policies `Repeating` and `FreqRep` perform significantly worse than the others. In more than half of all runs no reward is received at all. The median is taken over 10 000 runs.

4.2 θ as a tensor over state-action pairs

The performance of the learning algorithm for infinite reward parameter space is here tested on the `GeneralGridGame` environment. The experiments are designed to test how well $\pi^{\mathbf{R}}$ can approximate the optimal policy and how long it takes. $\mathbf{R}$ can never know for sure what the optimal policy is. The performance is measured in two ways. The first is by taking the fraction of actions in $\mathbf{R}$'s policy $\pi^{\mathbf{R}}$ that does not belong to the set of optimal actions $A^{\mathbf{R}}$. The second is by counting how often $\mathbf{H}$ and $\mathbf{R}$ are in the right position to receive a reward.

4.2.1 Comparison of assumptions

Just as when θ was a finite index, the effect of different assumptions are measured on the learning rate, see Section 4.1.2 for detailed explanations. The most important measurement is still that of `ApproxOpt`, since it is how a human is assumed to behave. However, this framework does not consider the optimality of $\mathbf{H}$, thus no incorrect assumptions will be tested. The experiments are run 100 times since the variance is quite low. The simulations lasts for 700 time steps to guarantee that all interesting learning behaviour is captured.

4.2.1.1 Different kinds of optimal human

The performance of the three different kinds of optimal policies and `ApproxOpt` can be seen in Figure 4.5 and Table 4.5. `IndeOpt` has again a very similar performance to that of `ApproxOpt`. `Optimal` and `Teaching` performs much better then the other policies. However, the version of `Teaching` constructed for this stetting failed to make any significant improvement on learning compared to `Optimal`. At first $\mathbf{R}$ has a random policy and on average about 70 % of the actions in that policy are non-optimal. After learning this value falls to about 35 % for the best policies and about 45 % for `ApproxOpt`.

The learning appear to follow a specific pattern each time. At first, in the initial 1-2 time steps, the learned policy gets slightly worse, to about 75 % error rate. This could be cased by the specifics of the game and that the policy is affected in all states, even those that are far away and little is known about. After that, the policy improves dramatically for a few time steps and then is gets worse again. Finally after a sort period of decreasing performance the learned policy continually improves before reaching a, somewhat, steady level.

4.2.1.2 Repeating policies

The result of the two repeating policies, `Repeating` and `FreqRep`, is here presented and compared to `ApproxOpt`, see Figure 4.6 and Table 4.6. `FreqRep` starts similarly to the others, but does not learn as much in the initial fast learning period. Instead, it slowly but steadily continues to learn during the whole simulation and reaches about the same level as `ApproxOpt` at the time limit.

`Repeating` learns about the same amount as `ApproxOpt` during the initial fast period. But, after that, when the performance for `ApproxOpt` levels out, `Repeating`

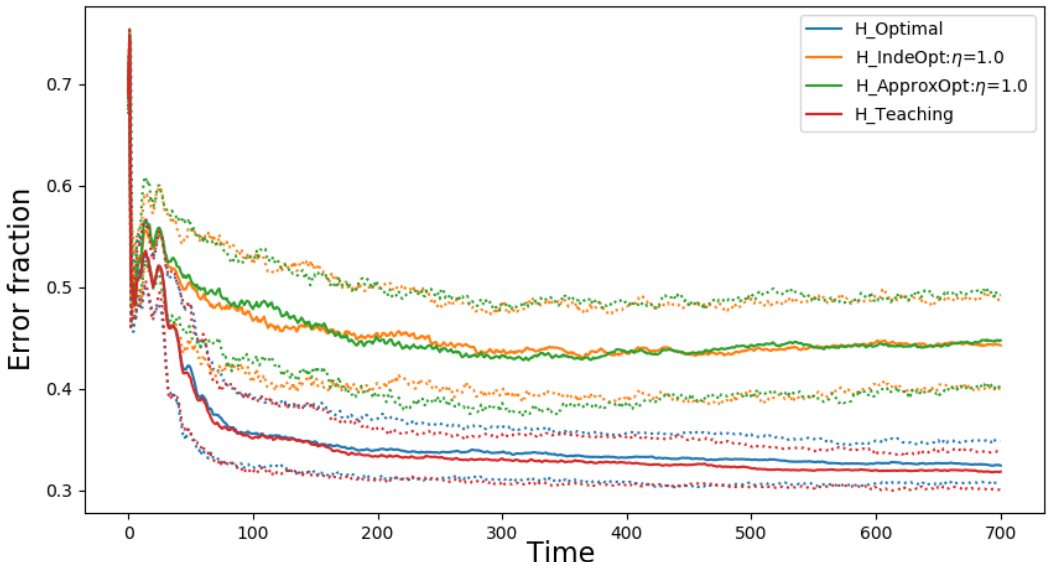


Figure 4.5: The solid lines show the fraction of actions in $\pi^{\mathbf{R}}$ that are not optimal actions. This is measured for the three different optimal policies, `Optimal`, `IndieOpt` and `Teaching`, as well as the default `ApproxOpt` policy. $\mathbf{R}$ assumes that $\mathbf{H}$ is following `ApproxOpt` with $\eta = 1.0$. The dotted lines show the average deviations between runs on both sides of the mean, i.e. the average result of all values lower or higher than the total average. `Teaching` and `Optimal` learns faster than `ApproxOpt` and `IndeOpt` which are very similar. Notice the similarities in the learning pattern for all policies. All values are averaged over 100 runs of 700 time steps.

Policy:	Optimal	IndeOpt	ApproxOpt	Teaching
H:	0.859	0.531	0.526	0.840
R:	0.725	0.455	0.448	0.730
Both:	0.725	0.322	0.314	0.730

Table 4.5: The average fraction of times that each agent is in the same column as the yellow square. The same is also measured for when both agents are in that column at the same time. If they both are, they receive a reward. Notice that Teaching and Optimal has a similar performance and they perform significantly better than the others. The average is taken over 100 runs.

keeps learning through the whole simulation and reaches a level similar to Teaching and Optimal at the end. Interestingly, if the simulation is run for long enough, both the repeating policies as well as Teaching and Optimal all level out at the same performance level which is an error rate of about 30 %. However, the repeating policies accumulate reward slower than all other policies.

Policy:	Repeating	FreqRep	ApproxOpt
H:	0.324	0.306	0.527
R:	0.230	0.184	0.454
Both:	0.097	0.095	0.319

Table 4.6: The average fraction of times that each agent is in the same column as the yellow square. The same is also measured for when both agents are in that column at the same time. If they both are, they receive a reward. Notice that both repeating policies accumulate very little reward. The average is taken over 100 runs.

4.2.2 Analysis of the learned θ

To see what **R** actually learns, the group of states which are attributed a reward of 1.0 is analyzed after a typical learning epoch of 700 time steps with the Teaching policy. The goal states, i.e. those where all three squares are in the same column, are in that group. Unfortunately, there are also some other states in there, states that **R** could not distinguish from the actual goal states. These are almost exclusively states where **H** is in the right place but **R** is one or two steps in front or behind, see Figure 4.7. The actions of **H** can likely not be used to infer any difference between them. Even Repeating suffers from this problem.

For those policies that are suboptimal, no state has an attributed reward of 1.0. The reason is that the rewards are approximated with averages, and a suboptimal action will have a value of less than 1.0. Therefore, a single mistake will reduce the average. However, those same states that are hard to differentiate and the correct states are almost always attributed the highest reward among all states.

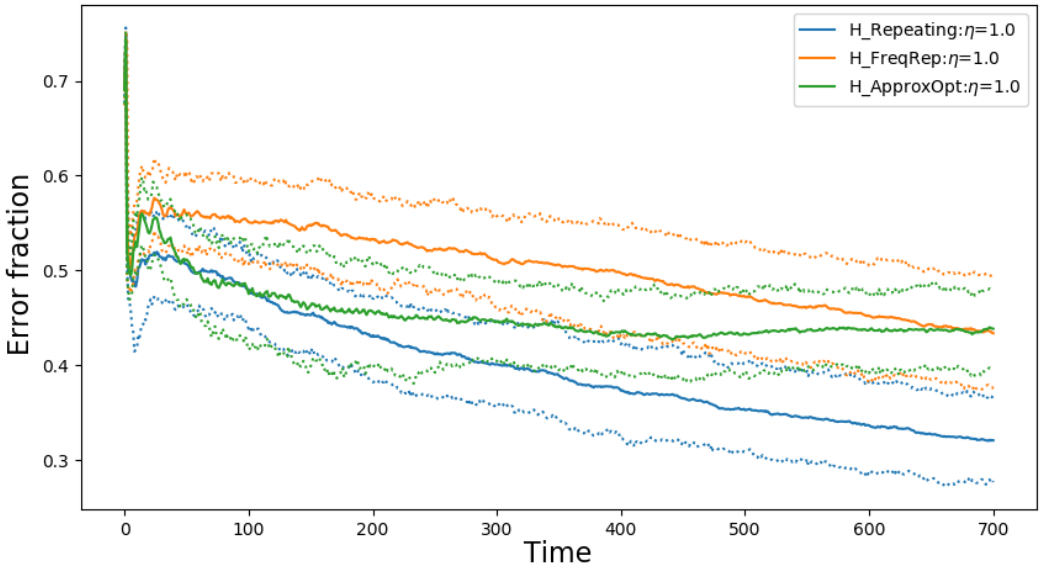
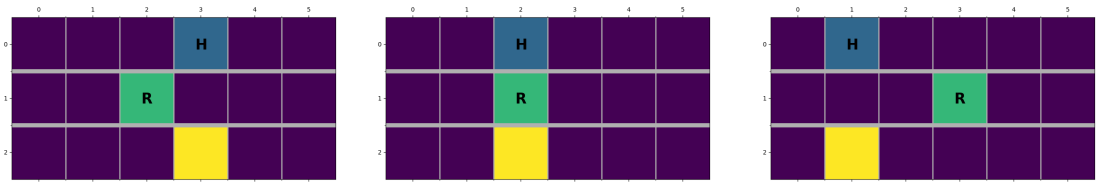


Figure 4.6: The solid lines show the fraction of actions in $\pi^{\mathbf{R}}$ that are not optimal actions. This is measured for the two different repeating policies, `Repeating` and `FreqRep`, along with the default `ApproxOpt` policy. $\mathbf{R}$ is aware of the repeating behaviour and does therefore use a modified likelihood, see Appendix A. The dotted lines show the average deviations between runs on both sides of the mean. Both repeating policies behave quite differently from the other policies. The period of fast learning in the beginning stops earlier for `FreqRep` than for other policies, but it does continue to improve for the entire training epoch and reaches roughly the same level as `ApproxOpt` at the end. `Repeating` has a similar period of fast learning as `ApproxOpt`, but the performance does not level out. Instead, it continues to learn, and after some time it it performs very well. All values are averaged over 100 runs.



(a) $\mathbf{R}$ is one step away from the correct position.

(b) $\mathbf{R}$ is in the correct position.

(c) $\mathbf{R}$ is two steps away from the correct position.

Figure 4.7: Three kinds of states that $\mathbf{R}$ often attribute a reward of 1.0 to. Only the states where all three squares are in the same column, (b), is supposed to have a reward of 1.0. This indicates that the difference in reward between them can not be inferred from the actions $\mathbf{H}$ makes.

5
Conclusion

This chapter discusses and summarizes the report. Potential explanations for observed phenomena are presented and the outcome of many design choices are analyzed. Necessary and possible further development is also presented. The summary contains the most important conclusions drawn from this project.

5.1 Discussion

Many topics are discussed and they are divided into three different sections. The first concerns the results and observations form the experiments. The second deal with the taken approach to achieve CIRL in the setting of this project. The final section concerns potential an necessary further development.

5.1.1 Results

The results show quite clearly that the way $\mathbf{H}$ behaves greatly affect the learning. $\mathbf{R}$ assumes that $\mathbf{H}$ will make mistakes, but the closer to optimal $\mathbf{H}$ behaves, the faster $\mathbf{R}$ learns. Even if $\mathbf{H}$ never makes any mistakes, learning can in some cases be improved by behaving in special ways that teaches $\mathbf{R}$ more (see definition of `Teaching`), as performing actions in different states can mediate different information. It is therefore important for $\mathbf{H}$ to occupy those states that mediate the most information. We see that regardless what assumption $\mathbf{R}$ has about the optimality of $\mathbf{H}$, $\eta^{\mathbf{H}}$, on average the performance will be the same after enough time. However, assuming a too high value for the optimality of $\mathbf{H}$ will result in an initial faster learning rate but also a larger variance between tries. The variance is troublingly large for all experiments. One reason is that $\mathbf{R}$ becomes too confident and is therefore not exploring new potential θs. This problem would have to be solved before the framework can be used in any applications. A starting point could be to always consider the number of observations that has been made and not only the current belief. This information could be used to avoid making a conclusion to early.

The policies used by $\mathbf{H}$ are not taken from the behavior of any actual person and may therefore not be representative. The results do however highlight that behaviour is important and that an AI can learn from a suboptimal human teacher. If $\mathbf{H}$ is capable of making mistakes, then it is important to consider that. But, if no mistakes are made, it may improve learning. The policy `Teaching` is not realistic, since it requires $\mathbf{H}$ to never make any mistakes. But, an approximately optimal teaching policy could be created which may be more realistic, and this should be

explored.

The games used for testing were designed to be challenging but not impossible. They were not meant to resemble any real world problems. Each framework was solely tested on one game, thus the results from the experiments can not be assumed to transfer to any other setting. There was an unforeseen consequence of using Q-values in the particular way of the simple setting (finite reward parameter space) and the design of `SimpleGridGame`. Namely that $\mathbf{R}$'s belief about the θs tend to equalize to the same probability unless they are constantly reinforced. Whether this is always a drawback is unclear, but in this case, it is not an advantage. The game design may also have caused the identical performance of `ApproxOpt` and `IndeOpt`, since they only di□er if there are a di□erent number of optimal actions in pair with $\mathbf{H}$'s actions.

The repeating policies does not perform very well in the simple setting. Surprisingly though, given enough time, they both learn as much as `Optimal` and `Teaching` in the general setting (infinite reward parameter space) if given enough time. For most policies $\mathbf{H}$ assumes that $\mathbf{R}$ will behave in the same way every time a state is visited. Therefore, $\mathbf{H}$ will also behave in the same way and the same information is obtained each time a state is visited. With the repeating policies however, $\mathbf{H}$ has di□erent assumptions about $\mathbf{R}$'s behaviour at di□erent times. It depends on what action $\mathbf{R}$ took in that state the previous times it was visited. Since $\mathbf{R}$'s policy is partially random, $\mathbf{H}$ will eventually have assumed all or many possible behaviours of $\mathbf{R}$. Each new assumption provides a new piece of information. It is not clear why repetition does not work as well in the simple setting. It could be since many times only a few states are ever visited as opposed to in `GeneralGridGame`, where the state is always changing regardless of what actions are taken. It is also not clear why `Teaching` performs better than `Optimal` in the general setting. Maybe the same logic for improving teaching in the simple setting does not hold in the general one.

5.1.2 Approach

A thorough search of the literature about IRL and CIRL yielded no instances of other researchers using Q-values to determine the intent of $\mathbf{H}$. One reason could be that the state transitions T are usually not known to the agent in RL. The Q-values can then not be calculated at the start. However, since CIRL focuses on other problems than learning the transitions, Q-values are well suited here. If it would be desirable to apply these frameworks to problems where the transitions are not known, then they could likely be combined with regular RL methods to learn the transitions simultaneously.

The framework for the general case has one specific advantage over the simplified one; It needs less meta parameters. It does not try to guess the optimality of the human and only uses one parameter for exploring. However, they both need to guess the discount factor of $\mathbf{H}$, which is an abstract concept to a human. The e□ect of incorrectly guessing the implicit discount factor of the human was not tested, but should be to guarantee the robustness of the framework.

For many real world scenarios it is not realistic to have a finite state space.

Continuous state spaces can be finitely discretized but some information does get lost in the process. There is no clear way of how to adapt the presented frameworks for the continuous cases but they can hopefully guide the development of ones that can, and this should be looked into in future research.

Since the framework for the general setting was not carefully analyzed, there exist some shortcomings. A minor one is that the reward only considers states, it should be possible to expand the methodology to cover the truly general case without too much effort. A larger problem is that states that $\mathbf{H}$ mostly moves towards, but is not actually interested in, are also attributed a high reward. If the humans level of optimality were to be considered also in the general case, that information could be used to statistically determine whether those difficult states are of interest. For the simple case, we can directly tell what $\mathbf{R}$ concludes about θ, specifically the probability that it is correct. In the general case, the measurement of interest is how closely approximated θ is. But, it is made more difficult since $\mathbf{R}$ does not know in what range the values of θ are in. This is why the performance is measured on the policy rather than θ. Another, possibly better, way of measuring the performance is to calculate the value of the policy, but this is quite time consuming.

q works well but there are likely much more efficient ways of drawing conclusions. It does take a long time to compute q and Q_θ, but since this only needs to be done once, before any training begins, it is not too problematic. Updating Q in the general case is probably not too problematic either. Since q changes very little between every time step, it does not take long to update Q. But, it depends on how long a time step is in an actual application and the available computing resources.

The policy `Repeating` performed very well in the general case, but it is not reasonable to expect a human to remember all actions that he or she has made previously. It is possible though, that a relaxed version of `Repeating` can be created. It may also not be that hard for humans to understand the teaching policy even if they can make mistakes. Combining those two policies may create an realistic and well performing policy. However, results like those by Palaniappan et al. (2017) can probably never be achieved with a suboptimal human.

5.1.3 Future work

The results and frameworks presented in this report is a good starting point, but there are still several parts that should be researched and improved. Both presented frameworks needs further development. Developing a framework that works with continuous state and action spaces would be interesting, potentially by using artificial neural networks. They could be used to approximate the Q-values and the q-values. It should also be tested how well $\mathbf{R}$ can take over both roles after having played with $\mathbf{H}$ for some time.

It is possible that very good results could be achieved by allowing the agents to communicate by a limited amount, where it should be investigated to see how burdensome it is for $\mathbf{H}$ to give $\mathbf{R}$ some guidance. And also if direct guidance can be given without increasing the potential risk of causing Value Alignment problems.

5.2 Summary

This project has managed to take some initial steps into the field of Cooperative inverse reinforcement learning with a suboptimal human agent. The problems faced in this area were discussed and potential solutions were presented. One such problem is how to model the behaviour of a human agent. A few different models were developed and their realism was discussed. One of these models was then selected to be the foundation for the development of two CIRL frameworks for slightly different scenarios. These two frameworks were analyzed and then tested on developed benchmark games. Tests were made and compared for all developed human agent models. The results clearly show learning and some interesting behaviours of the frameworks. It is also evident that the behaviour of the human agent has a large affect on learning. Even though the human is not expected to behave optimally, the AI learns faster if the human does. If there are multiple optimal policies, then they can have different teaching capability. The results and insights from this project are very interesting, and it is clear that more research is needed.

References

Abbeel, P., Dolgov, D., Ng, A. Y., & Thrun, S. (2008). Apprenticeship learning for motion planning with application to parking lot navigation. In *Intelligent robots and systems, 2008. iros 2008. ieee/rsj international conference on* (pp. 1083–1090).

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on machine learning* (p. 1).

A.E.S. (2016). Why firms are piling into artificial intelligence. *The Economist*.

Armstrong, S., & Leike, J. (2016). *Towards interactive inverse reinforcement learning.* Retrieved from https://jan.leike.name/publications/Towards%20Interactive%20Inverse%20Reinforcement%20Learning%20-%20Armstrong,%20Leike%202016.pdf

Carey, R. (2017). Incorrigibility in the cirl framework. *arXiv preprint arXiv:1709.06275*.

Hadfield-Menell, D., Russell, S. J., Abbeel, P., & Dragan, A. (2016). Cooperative inverse reinforcement learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems 29* (pp. 3909–3917). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/6420-cooperative-inverse-reinforcement-learning.pdf

Jensen, M. C., & Meckling, W. H. (1976). Theory of the firm: Managerial behavior, agency costs and ownership structure. *Journal of financial economics*, *3*(4), 305–360.

Lin, X., Beling, P. A., & Cogill, R. (2014). Multi-agent inverse reinforcement learning for zero-sum games. *arXiv preprint arXiv:1403.6508*.

Palaniappan, M., Malik, D., Hadfield-Menell, D., Dragan, A., & Russell, S. (2017). E□cient cooperative inverse reinforcement learning.

Russel, S., & Norvig, P. (2010). *Artificial intelligence* (3rd ed.). Upper Saddle River, New Jersey 07458: Pearson Education, Inc.

Russo, D., Roy, B. V., Kazerouni, A., & Osband, I. (2017). A tutorial on thompson sampling. *CoRR*, *abs/1707.02038*. Retrieved from http://arxiv.org/abs/1707.02038

Shead, S. (2017). Google deepmind's alphago ai beat the best go player in the world in its first game. *Business insider*.

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction*. MIT press.

A
Policies

A.1 Policies for H

The definitions of all policies and the assumptions they use are presented here.

1.1 & 2.1 `Optimal`: $\mathbf{H}$ plays any optimal action with uniform probability.

$$\pi^H \quad U(A^H) \tag{A.1}$$

1.1 & 2.2 `ApproxOpt`: $\mathbf{H}$ probabilistically plays an action based on the softmax of that action's Q-value paired with the best joint a^R.

$$\pi^H \quad P(a^H) := \frac{\exp\Big(\eta Q_\theta\ (s, a^H, a^R)\Big)}{\sum_{a^H\ \ A^H} \exp\Big(\eta Q_\theta\ (s, a^H, a^R)\Big)},\ (a^H, a^R) \quad A \tag{A.2}$$

1.1 & 2.3 `Teaching`: Similar to `Optimal` but avoids moving towards the goal if there exists other optimal actions.

1.2 & 2.2 `IndeOpt`: Similar to `ApproxOpt` but $\mathbf{H}$ now assumes that $\mathbf{R}$ may not play the action that is optimal in pair with a^H. Instead $\mathbf{R}$ may play any optimal action. Here A^R is dependent on a^H and s.

$$\pi^H \quad P(a^H) := \frac{\exp\Big(\frac{\eta}{|\ A\ |} \sum_{a^R\ \ A^R} Q_\theta\ (s, a^H, a^R)\Big)}{\sum_{a^H\ \ A^H} \exp\Big(\frac{\eta}{|\ A\ |} \sum_{a^R\ \ A^R} Q_\theta\ (s, a^H, a^R)\Big)} \tag{A.3}$$

1.3 & 2.2 `Repeating`: Similar to `ApproxOpt` but $\mathbf{R}$ is assumed to repeat the last action in this specific state. a_p^R is the action selected by $\mathbf{R}$ the previous time s was visited.

$$\pi^H \quad P(a^H) := \frac{\exp\Big(\eta Q_\theta\ (s, a^H, a_p^R)\Big)}{\sum_{a^H\ \ A^H} \exp\Big(\eta Q_\theta\ (s, a^H, a_p^R)\Big)} \tag{A.4}$$

1.4 & 2.2 `FreqRep`: Similar to `ApproxOpt` but R is assumed to stochastically select an action based on the observed selection frequency. $\nu(s, a^R)$ is the fraction of times a^R has been chosen in s.

$$\pi^H \quad P(a^H) := \frac{\exp\Big(\eta \sum_{a^R\ \ A^R} Q_\theta\ (s, a^H, a^R)\nu(s, a^R)\Big)}{\sum_{a^H\ \ A^H} \exp\Big(\eta \sum_{a^R\ \ A^R} Q_\theta\ (s, a^H, a^R)\nu(s, a^R)\Big)} \tag{A.5}$$

A.2 Likelihoods when θ is a finite index

Since $\mathbf{R}$ always assumes that $\mathbf{H}$ is following the ApproxOpt policy, except for the repeating policies, only the repeating policies needs a different likelihood function.
ApproxOpt:

$$P_\theta(a_t^H \mid s_t) := \frac{\exp\left(\eta Q_\theta(s_t, a_t^H, a^R)\right)}{\sum_{a^H \ A^H} \exp\left(\eta Q_\theta(s_t, a^H, a^R)\right)}. \tag{A.6}$$

Repeating:

$$P_\theta(a_t^H \mid s_t) := \frac{\exp\left(\eta Q_\theta(s, a^H, a_p^R)\right)}{\sum_{a^H \ A^H} \exp\left(\eta Q_\theta(s, a^H, a_p^R)\right)} \tag{A.7}$$

FreqRep:

$$P_\theta(a_t^H \mid s_t) := \frac{\exp\left(\eta \sum_{a^R \ A^R} Q_\theta(s, a^H, a^R)\nu(s, a^R)\right)}{\sum_{a^H \ A^H} \exp\left(\eta \sum_{a^R \ A^R} Q_\theta(s, a^H, a^R)\nu(s, a^R)\right)} \tag{A.8}$$

A.3 θ as a matrix over states

The default update and action selection as well as those for the repeating policies.

A.3.1 Updating beliefs for repeating policies

Since $\mathbf{R}$ always assumes that $\mathbf{H}$ is following the ApproxOpt policy except for the repeating policies, only the repeating policies needs a different update.

ApproxOpt:

$$\begin{aligned} a_t(s) &= \underset{a^H \ A^H, a^R \ A^R}{\text{minimum}} \left\{ q\ (s_t, a^H, a^R, s) \right\} \\ b_t(s) &= \underset{a^H \ A^H, a^R \ A^R}{\text{maximum}} \left\{ q\ (s_t, a^H, a^R, s) \right\} \\ v_t(s) &= \underset{a^R \ A^R}{\text{maximum}} \left\{ q\ (s_t, a_t^H, a^R, s) \right\} \end{aligned} \tag{A.9}$$

Repeating:

$$\begin{aligned} a_t(s) &= \underset{a^H \ A^H}{\text{minimum}} \left\{ q\ (s_t, a^H, a_p^R, s) \right\} \\ b_t(s) &= \underset{a^H \ A^H}{\text{maximum}} \left\{ q\ (s_t, a^H, a_p^R, s) \right\} \\ v_t(s) &= q\ (s_t, a_t^H, a_p^R, s) \end{aligned} \tag{A.10}$$

FreqRep:

$$\begin{aligned} a_t(s) &= \underset{a^H \ A^H}{\text{minimum}} \left\{ \sum_{a^R \ A^R} q\ (s_t, a^H, a^R, s)\nu(s, a^R) \right\} \\ b_t(s) &= \underset{a^H \ A^H}{\text{maximum}} \left\{ \sum_{a^R \ A^R} q\ (s_t, a^H, a^R, s)\nu(s, a^R) \right\} \\ v_t(s) &= \sum_{a^R \ A^R} q\ (s_t, a_t^H, a^R, s)\nu(s, a^R) \end{aligned} \tag{A.11}$$

A.3.2 Action selection for repeating policies

The default policies that $\mathbf{R}$ follows and the modified versions for repeating policies.

`ApproxOpt`:

$$\pi^R(s_t) := \underset{a^R \ A^R}{\text{argmax}}\left\{\underset{a^H \ A^H}{\text{maximum}}\left\{Q_t(s_t, a^H, a^R)\right\}\right\} \tag{A.12}$$

`Repeating`:

$$\pi^R(s_t) := \underset{a^R \ A^R}{\text{argmax}}\left\{Q_t(s_t, a^H, a^R),\ a^H = \underset{a^H \ A^H}{\text{argmax}}\left\{Q_t(s_t, a^H, a_p^R)\right\}\right\} \tag{A.13}$$

`FreqRep`:

$$\pi^R(s_t) := \underset{a^R \ A^R}{\text{argmax}}\left\{\underset{a^H \ A^H}{\text{maximum}}\left\{Q_t(s_t, a^H, a^R)\nu(s, a^R)\right\}\right\} \tag{A.14}$$

B

Code

The code used in this project can be found in the GitHub repository at `https://github.com/cajoek/CIRL`.