CHALMERS

Learning to Play Games from Multiple Imperfect Teachers

Master's Thesis in Complex Adaptive Systems

JOHN KARLSSON

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2014

Learning to Play Games from Multiple Imperfect Teachers

John Karlsson

Examiner: CHRISTOS DIMITRAKAKIS

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering

Göteborg, Sweden June 2014

Abstract

This project evaluates the modularity of a recent Bayesian Inverse Reinforcement Learning approach [1] by inferring the sub-goals correlated with winning board games from observations of a set of agents. A feature based architecture is proposed together with a method for generating the reward function space, making inference tractable in large state spaces and allowing for the combination with models that approximate state-action values. Further, a policy prior is suggested that allows for least squares policy evaluation using sample trajectories. The model is evaluated on randomly generated environments and on Tic-tac-toe, showing that a combination of the intentions inferred from all agents can generate strategies that outperform the corresponding strategies from each individual agent.

Contents

1

Introduction

Several board games have remained as important targets in the field of Artificial Intelligence for being well defined environments with simplistic sets of rules, yet still being unsolved after decades of research. In the recent years, the surge of *big data* applications have made *unsupervised learning* techniques increasingly popular. Many games have extensive and growing databases of expert play available, and this has become an interesting source for knowledge extraction.

Multitask inverse reinforcement learning is concerned with inferring the motivations of different agents trying to solve the same task in a dynamic environment. The solutions that the agents choose are biased by their different preferences. By studying databases of game records, it is possible to infer the preferences that motivate the agents' behaviour.

This work focuses on inferring the motivations of multiple imperfect teachers whose preferences have formed during interaction with the complex and adaptive system represented by a community of co-evolving agents. Over time, the game playing agents reach equilibrium where their sets of preferences no longer change. This makes reinforcement learning techniques applicable, which generally assume a stationary, Markovian environment.

1.1 Problem formulation

The benefits of board games is that values can be assigned to any terminal state (final position) in terms of a win, loss, draw, or by a scoring mechanism defined by the game rules. Via backward induction, the expected value for any state in the game can be calculated. This allows for the construction of an optimal strategy that uses one step lookahead to choose the action (move) that maximises the expected value of the next state. However, due to time and space constraints, the value calculation is generally infeasible. A few notable examples of game state-space complexities[1] are Go (10^{171}), Chess (10^{47}), Havannah (10^{127}), Hex (10^{57}) and Othello (10^{28}). The fact that definite goals exist only as terminal states also becomes the largest difficulty.

By utilising demonstrations made by experts in the game, it is possible to perform inference on their intrinsic sub-goals that correlate with winning outcomes. Unfortunately, none of the experts are infallible, and any statistical approaches used must take sub-optimality, even with regards to the experts own intentions, into account. In fact, in a recent Bayesian approach [1], this is attempted for general environments that are not necessarily board games, by repeatedly solving for the optimal strategy given different proposed sub-goals, to be formalised later.

In large state spaces, clever methods must be incorporated to make such approaches feasible. Also, in board games, it is assumed that all experts share some common goals, which converge to one single goal near terminal states. The purpose of this project is to implement the algorithm in [1] and to extend it to large state spaces and to further utilise the structure of game trees to make the fully Bayesian approach feasible for inferring experts' intentions.

1.2 Motivation from related fields

One way of transferring knowledge is by demonstrations, such as in the field of Programming by Demonstration (PbD) [3] where an engineer may demonstrate to a robot how to perform a task, rather than having to perform the often hard and time consuming work of engineering the behaviour by hand. This concept allows for agents to request demonstrations and exchange advice between each other, combining it with individual experiences acquired through interaction with the environment [4]. The goal of *transfer learning* is to share knowledge learned within and between different domains [5]. As an example, we might want to perform training in one domain while all the training data resides in another domain with a different feature space and distribution; e.g. in another agent's memory of experiences. A motivation in this field is the need for machine learning methods to retain and reuse knowledge [5].

An agent may have have cognitive or emotional limitations of information processing that limits its ability to act and plan perfectly according to its own preferences. The field of *preference elicitation* is concerned with extracting the preferences of an agent for the purpose of constructing decision support systems such as recommender systems and personalised marketing systems [6]. *Utility* based approaches have been made to link the theory to applications such as (inverse) reinforcement learning [7] and e-commerce recommender systems [8].

A difficulty in learning to play games is that it usually requires a long sequence of moves before any definite success or failure can be determined, often referred to as the *credit assignment problem* [9]. When children learn a task, they quickly manage to deduce which actions correlate with a positive outcome, and consequently proceed

[1]State-space complexity — The number of positions reachable from the initial position of the game [2].

to take those actions more often. The task of dopamine neurons is to associate stimuli in the environment with bodily needs [10]; this is the biological solution to the credit assignment problem. When making faulty predictions, the dopamine neurons react by propagating the information about the presence or absence of a reward. This form of message passing is a way of learning to predict the timing and magnitude of future rewards [11], which has a close resemblance to *temporal difference learning* — one of the most central classes of *reinforcement learning* algorithms [12, 13].

2

Theory

Reinforcement Learning is an area of machine learning where an *agent* acting in a (potentially unknown) *environment* must learn an optimal strategy in a feedback loop of signals transmitted between them. The agent's input from the environment are (1) reinforcement signals referred to as *rewards* and (2) sensory information about current *state* that the environment is said to be in. The agent's output signals are referred to as *actions*.

Although there is a notion of rewards, the reinforcement learning problem is different from supervised learning problems in the following sense. Improving knowledge about optimal actions in stochastic environments entails trial and error via multiple interactions. This carries the problem of balancing between *exploration* — improving the certainty about current beliefs, and *exploitation* — taking those actions that have been found to be effective in the past.

In a machine learning setting, the agent's interaction with the environment is modeled as a *Markov decision process* (MDP) (Section 2.1.1), where it is to maximize an inherent and to the agent subjective *utility* (Section 2.1.2) which is a function of the rewards.

Also, the actions may affect not only immediate rewards, but all subsequent rewards as well. In many cases, the rewards are delayed until several actions have been made, leading to the *credit assignment problem* where the agent must learn which actions are to be assigned credit or blame for the final outcome (see Section 2.3).

In *Inverse reinforcement learning* (Section 2.4.1), the agent is referred to as an *expert* or demonstrator of a certain *task* (modeled as an MDP) and the problem is to infer the parameters of the environment (i.e. the reinforcement learning problem) as the agent perceives it. In short, the problem is to infer the agent's motivations (utility) given some data generated by the agent.

2.1 Decision making under uncertainty

This section treats decision making in Markov decision processes and the formalisation of agent strategies.

Decision theory is concerned with the optimisation of *utility*: a subjective measure inherent in the decision maker. When faced with a decision under uncertanty, *expected utility* is defined as the sum of all possible outcomes weighed by their relative likelihoods (which may also be subjective), acting as a basis framework for the formalisation of *rationality* in agents.

2.1.1 Markov decision processes

A *Markov decision process* (MDP) is a Markov process[1] whose transitions are conditioned on a decision — the agent's action — made in every state. It is defined by a tuple $\mu = (\mathcal{S},\mathcal{A},\mathcal{T},\rho)$ whose elements are described as follows.

$\mathcal{S}$ is a set of states which may be either discrete or continuous. In every time step t, the Markov process is in a state $s_t \in \mathcal{S}$, and transitions to a state $s_{t+1} \in \mathcal{S}$. Although the focus will be exclusively on discrete state spaces, the size will still be too large for practical enumeration in the algorithms considered here (Section 2.2). This calls for a *feature representation* of the state space (Section 2.3.1), which is a method that easily translates to continuous state spaces as well.

$\mathcal{A}$ is the set of actions, on which the stochastic transitions of the MDP are conditioned. Like the state space, the set of actions may also be continuous, for example when the action represents the amount of force to apply in a control problem. All methods presented here will however assume that there is a small, discrete set of actions available in each state.

The transition kernel $\mathcal{T} \triangleq \{\tau_a(\cdot \mid s) : s \in \mathcal{S}, a \in \mathcal{A}\}$ defines a set of $|\mathcal{S} \times \mathcal{A}|$ functions for the transition probabilities from state s to successor states s' given action a, s.t. $\forall (s,a) \in \mathcal{S} \times \mathcal{A} : \sum_{s' \in \mathcal{S}} \tau_a(s' \mid s) = 1$. A convenient alternative representation with many practical advantages from linear algebra is to define $\mathcal{T}$ as a set of $|\mathcal{A}|$ matrices $\tau_a(s' \mid s) \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$.

A reward function $\rho : \mathcal{S} \to \mathbb{R}$ defines the numerical reward $\rho(s)$ generated by the transition *to* state $s \in \mathcal{S}$.[2]

Example MDP

A deterministic mapping $\pi : \mathcal{S} \to \mathcal{A}$ from states to actions results in a single strategy-induced transition matrix $\boldsymbol{\mathcal{T}}^{\pi}$ which defines the kernel of a regular Markov chain. As an example of the dynamics of a simple MDP with 3 states and 2 actions, Figure 2.1 shows a pictorial representation of the two different Markov chains induced by the two

[1]Markov process — A stochastic process which has the *Markov property*, i.e. being memoryless, s.t. transitions between states depend solely on the current state that the system is in.

[2]Sometimes more general definitions are used, such as the reward function $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ defined on the space of all possible transitions $s_t \xrightarrow{a_t} s_{t+1}$.

strategies $\pi_1(\cdot) = a_1$ and $\pi_2(\cdot) = a_2$. In this example, $\mathcal{T}^{\pi_1}$ and $\mathcal{T}^{\pi_2}$ are given by:

$$\mathcal{T}^{\pi_1} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad \mathcal{T}^{\pi_2} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix} \tag{2.1}$$

s.t. element (s_t, s_{t+1}) denotes the transition probabilities between those states. Thus, $\mathcal{T}^{\pi}$ for any particular strategy π is then a mixture of the two.

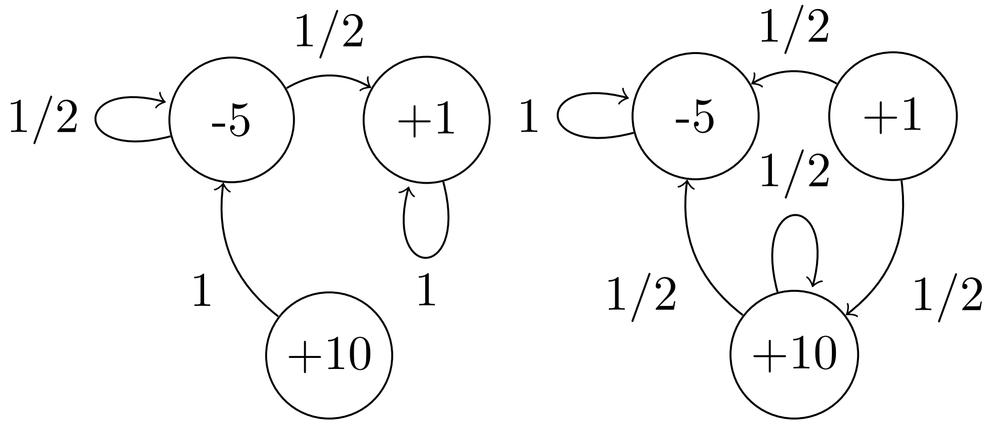


Figure 2.1: A pictorial representation of an MDP where arrows indicate transition probabilities for a_1 (left) and a_2 (right). Rewards are shown inside the nodes and are supposed to be identical in the two images.

2.1.2 Policies

To be able to formulate an optimal *policy*, i.e. an optimal strategy for an agent acting in an MDP, it is necessary to make assumptions regarding the objective function being optimised.

Since the preferences of an agent are indeed subjective, decision theory handles the problem by formalising the notion of an agent's *utility* as a function $U : \mathcal{R} \rightarrow \mathbb{R}$, where $\mathcal{R}$ can formally be seen as the set of all possible reward sequences of different lengths. It is then said that if an agent prefers a reward r_1 to r_2, then $\mathbb{E}\big[U(r_1)\big] > \mathbb{E}\big[U(r_2)\big]$; also known as the *expected utility hypothesis* [14].

Further, the following assumption is a common one to make, which is found in most literature on MDPs [15].

Assumption 1 (Agent utility). *The agent acts to maximize the discounted sum of future rewards*

$$U_t \triangleq \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}, \tag{2.2}$$

where $\gamma \in (0,1]$ is a discount factor and $r_t = \rho(s_t)$ is the deterministic reward received at time t upon reaching state s_t.

This is an important assumption in Inverse Reinforcement Learning (Section 2.4.1) when observing an agent that acts in an environment since it connects preference elicitation with the problem of inferring $\rho \mid \boldsymbol{D}$ for some agent demonstrations $\boldsymbol{D}$.

Definition 1 (Policy)**.** *A stationary policy $\pi \in \mathcal{P}$ defines a probability distribution over the set of actions given a particular state:*

$$\pi(a \mid s) = \mathbb{P}(a_t = a \mid s_t = s), \qquad a \in \mathcal{A}, s \in \mathcal{S} \tag{2.3}$$

which can be seen as a strategy used by an agent acting in an MDP.

The most general form of policies are those that are stochastic and history-dependent [16]; whereas the most specific are stationary deterministic policies. A *stationary* policy is one that is time-invariant, i.e. its memoryless distribution over actions depends only on the current state, and not on the actions or observations made in previous time steps:

$$\pi(a \mid s_t, s_{t-1}, \dots, s_0) = \pi(a \mid s_t), \qquad a \in \mathcal{A} \tag{2.4}$$

For the calculation of optimal policies (Section 2.2) in *discounted infinite-horizon* MDPs, it is sufficient to consider only such stationary policies (see [16] pp. 152), since the optimality conditions imply the existence of an optimal stationary policy. However, since board games are *undiscounted* and *finite-horizon* MDPs, the policy evaluation algorithms reviewed in Section 2.3.1 are restricted to policies that are *proper*, meaning that they are guaranteed to reach a terminal state.

2.2 Optimality

To construct optimal policies it is needed to evaluate a policy based on the expected utility hypothesis. The objective function is the expectation of the assumed utility in (2.2) given some starting state $s_t = s$:

$$\mathbb{E}[U_t \mid s_t = s] \equiv \mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \;\middle|\; s_t = s\right].$$

The joint dynamics of μ and π are then used to enumerate all possible transitions on which the reward function $\rho(s_t \xrightarrow{a_t} s_{t+1}) = \rho(s_{t+1})$ is defined. This leads to the following recursive formulation where $\mathbb{E}[U_t]$ is defined in terms of $\mathbb{E}[U_{t+1}]$:

$$\mathbb{E}_\mu^\pi\big[U_t \mid s_t = s\big] = \sum_{s' \in \mathcal{S}} \mathbb{P}(s_{t+1} = s' \mid s_t = s)\Big(\rho(s') + \gamma \mathbb{E}_\mu^\pi\big[U_{t+1} \mid s_{t+1} = s'\big]\Big), \tag{2.5}$$

where the indices μ and π indicate that the expectation is over the dynamics of the MDP $s_{t+1} \mid s_t \sim \tau_{a_t}$ and the actions of the policy $a_t \mid s_t \sim \pi$ s.t.

$$\begin{aligned}\mathbb{P}(s_{t+1} \mid s_t) &= \sum_{a \in \mathcal{A}} \mathbb{P}(s_{t+1} \mid s_t, a_t = a)\mathbb{P}(a_t = a \mid s_t) \\ &\equiv \sum_{a \in \mathcal{A}} \tau_a(s_{t+1} \mid s_t)\pi(a_t = a \mid s_t).\end{aligned}$$

This is a good time to introduce the *value function* which serves a more intuitive notation:

$$\begin{aligned}V^\pi_\mu(s) &\triangleq \mathbb{E}^\pi_\mu\big[U_t \mid s_t = s\big] \\ &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(a \mid s)\tau_a(s' \mid s)\Big(\rho(s') + \gamma V^\pi_\mu(s')\Big),\end{aligned} \tag{2.6}$$

From here on, the subscript μ will be dropped whenever it is clear from context. Later, however, when the need arises to distinguish between the values in different tasks (MDPs), the only varying element will be $\rho \in \mu$ and the notation V^π_ρ will be used instead. An optimal policy π^* has the property $V^{\pi^*} = V^*$ where

$$V^*(s) \triangleq \max_a \left\{ \sum_{s' \in \mathcal{S}} \mathbb{P}_a(s' \mid s)\Big(\rho(s') + \gamma V^*(s')\Big) \right\}, \qquad \forall s \in \mathcal{S} \tag{2.7}$$

Another common notation is the *Q-value* of a state-action pair (s,a), defined as the expected utility conditioned on action a taken in state s:

$$Q^\pi(s,a) \triangleq \mathbb{E}\big[U \mid s,a\big] \tag{2.8}$$

$$= \mathbb{E}\big[\rho(s') \mid s,a\big] + \gamma \sum_{s' \in \mathcal{S}} \tau_a(s' \mid s) \sum_{a' \in \mathcal{A}} \pi(a' \mid s)Q(s',a') \tag{2.9}$$

$$= \sum_{s' \in \mathcal{S}} \tau_a(s' \mid s)\Big(\rho(s') + \gamma V^\pi(s')\Big) \tag{2.10}$$

and $$Q^*(s,a) \triangleq Q^{\pi^*}(s,a), \tag{2.11}$$

s.t. $$V^*(s) \equiv \max_a \Big\{ Q^*(s,a) \Big\} \tag{2.12}$$

2.2.1 Bellman Optimality Equations

The value of a policy in (2.6) and that of the optimal policy in (2.7) are the two equations known as the *Bellman equation* and *Bellman optimality equation* respectively [17], which form the basis of the *policy evalutation* and *value iteration* algorithms. These algorithms both apply the respective equation as an update rule to all possible states until the values of all states no longer change.

Consider the MDP from the previous example in Figure 2.1 and assume an infinite horizon and $\gamma = \frac{1}{2}$ for simplicity, and focus on the left side induced by the policy $\pi_1(\cdot) = a_1$. The value $V^{\pi_1}(s_2)$ of the top right state denoted by s_2 for this policy is the easiest to calculate, since it is given by the geometric series $V^{\pi_1}(s_2) = 1 \times (\rho(s_2) + \gamma V^{\pi_1}(s_2)) = 2$. By backtracking to the top left state, denoted by s_1, there is only one unknown variable (value) which can be solved by $V^{\pi_1}(s_1) = \frac{1}{2} \times (-5 + \gamma V^{\pi_1}(s_2)) + \frac{1}{2} \times (1 + \gamma V^{\pi_1}(s_1))$, giving $V^{\pi_1}(s_1) = -2$. Similarly, $V^{\pi_1}(s_3) = -6$.

At a more general note, Equation (2.6) forms a linear system of equations:

$$\boldsymbol{v}^{\pi} = \boldsymbol{\mathcal{T}}^{\pi}\boldsymbol{r} + \gamma\boldsymbol{\mathcal{T}}^{\pi}\boldsymbol{v}^{\pi}, \tag{2.13}$$

which has the solution given by the *policy evaluation* step:

$$\boldsymbol{v}^{\pi} = (\boldsymbol{I} - \gamma\boldsymbol{\mathcal{T}}^{\pi})^{-1}\boldsymbol{\mathcal{T}}^{\pi}\boldsymbol{r}. \tag{2.14}$$

In the case discussed above, the value function is $\boldsymbol{v}^{\pi_1} = (-2, 2, -6)^\top$. In other words, given the policy π_1 that only chooses action a_1, the bottom state is not very good. A *policy improvement* step would update the policy similarly to Equation (2.7):

$$\begin{aligned} \pi_{n+1}(s) &= \arg\max_a \left\{ \sum_{s' \in \mathcal{S}} \mathbb{P}_a(s' \mid s)\Big(\rho(s') + \gamma V_n(s')\Big) \right\}, \quad \forall s \in \mathcal{S} \\ &\Leftrightarrow \\ \pi_{n+1} &= \arg\max_\pi \big\{ \boldsymbol{\mathcal{T}}^{\pi}(\boldsymbol{r} + \gamma \boldsymbol{v}_n) \big\}, \end{aligned} \tag{2.15}$$

where the subscript n denotes the iteration index. In this case, the updated policy would be $\pi_{n+1}((s_1, s_2, s_3)^\top) = (a_1, a_1, a_2)^\top$, i.e. where the action changed only in s_3 from a_1 to a_2, giving $\boldsymbol{v}^{\pi_{n+1}} = (-2, 2, \frac{8}{3})^\top$. The policy induced transition kernel $\boldsymbol{\mathcal{T}}^{\pi_{n+1}}$ is then constructed by choosing rows 1,2 from $\boldsymbol{\mathcal{T}}^{\pi_1}$ and row 3 from $\boldsymbol{\mathcal{T}}^{\pi_2}$. In yet another and final iteration in the same manner, the algorithm would converge to the optimal policy denoted by $\pi^* = \pi_{n+2}((s_1, s_2, s_3)^\top) = (a_1, a_2, a_2)^\top$, giving the optimal value function $\boldsymbol{v}^* = \boldsymbol{v}^{\pi^*} = (2, \frac{8}{3}, \frac{8}{3})^\top$. This concludes the example of the *policy iteration* algorithm.

Policy iteration iteratively performs steps (2.14) and (2.15) until convergence, i.e. until $\|\boldsymbol{v}_n - \boldsymbol{v}_{n+1}\| \le \varepsilon$ (which is also implied when $\pi_{n+1} = \pi_n$).

Value iteration combines these steps such that the policy is not queried, but the maximum over actions is taken in each iteration, see Algorithm 1.

2.3 Temporal Difference learning

Temporal Difference (TD) learning methods [12, 18] attempt to estimate the value function $V^{\pi}(\cdot)$ for a given policy π:

$$V^{\pi}(s_t) = \rho(s_{t+1}) + \gamma V^{\pi}(s_{t+1}), \tag{2.16}$$

given one or more finite sample trajectories of kind $\{s_t\}_{t=1}^{T}$ generated by π, where s_T is a terminal state with a reward (outcome) $\rho(s_T)$. They attempt to solve the (temporal) *credit assignment problem* inherent in problems with delayed rewards [9], such as board games where the reinforcement is usually delayed until the very last (terminal) states.

Algorithm 1: Value Iteration

```
Input: MDP $\mu$ Discount factor $\gamma$, Precision parameter $\varepsilon$
Result: Vector $\boldsymbol{v}$ s.t. $v_s = V^*_\mu(s), \forall s \in \mathcal{S}$
$\boldsymbol{v}_0 \leftarrow \mathbf{0}$
$n \leftarrow 0$
repeat
    $n \leftarrow n + 1$
    foreach $s \in \mathcal{S}$ do
        $\boldsymbol{v}_n(s) \leftarrow \max_a \left\{ \sum_{s' \in \mathcal{S}} \tau_a(s' \mid s) \Big( \rho(s') + \gamma \boldsymbol{v}_{n-1}(s') \Big) \right\}$
    end
until $\|\boldsymbol{v}_n - \boldsymbol{v}_{n-1}\| \leq \varepsilon$;
return $\boldsymbol{v}_n$
```

The method attempts to match current estimates $V(s_t)$ with (hopefully) more accurate beliefs about the values for future states $V(s_k)$, $k > t$. This takes advantage of the assumption that subsequent values $V(s_t), V(s_{t+1}), \ldots$ are correlated.

A regular Monte Carlo (MC) learning method would approximate the value $V(s_t)$ by generating a rollout[3] sample U_t and applying the update rule:

$$\widehat{V}(s_t) \leftarrow \widehat{V}(s_t) + \alpha\big(U_t - \widehat{V}(s_t)\big), \tag{2.17}$$

where α is a step size parameter. Clearly, if the current prediction $\widehat{V}_t$ matches the *target* value $\mathbb{E}[U_t]$, the value function has been learned and only fluctuates depending on the step size α. The issue with this method, however, is that a full trajectory needs to be generated for every sample of U_t. TD-methods instead attempt to minimize the *TD-error* between subsequent states by applying the update rule:

$$\widehat{V}(s_t) \leftarrow \widehat{V}(s_t) + \alpha\big(\rho(s_{t+1}) + \gamma\widehat{V}(s_{t+1}) - \widehat{V}(s_t)\big). \tag{2.18}$$

The target instead becomes $\mathbb{E}\big[\rho(s_{t+1}) + \gamma V(s_{t+1})\big]$, which, since $\widehat{V}(s_{t+1})$ is itself a current estimate maintained by the algorithm, makes TD-learning a *bootstrapping* method.

Algorithm 2 presents a version of this method which queries the policy in an on-line fashion, updating the value function as the trajectory is being generated.

For the coming section, it is instructive to use the state-action value notation $Q^\pi(s,a)$ and to briefly introduce *on-policy* and *off-policy* learning, for which two common examples (among TD-methods) are *Sarsa* [12, 19] and *Q-learning* [20] respectively. Both of these methods attempt to learn an optimal value function, whereas the former of the two works nearly identically to Algorithm 2 apart from two modifications. First, to choose actions a_t,$a_{t+1}, \ldots$, Sarsa queries a policy $\pi_{\boldsymbol{Q}^\pi}$ that chooses its actions based on the

[3]Rollout - sample trajectory from a starting state following some policy π until a terminal state is reached, resulting in a sampled cumulative sum of returns U_t.

Algorithm 2: TD-learning of V^π

```
Input: MDP $\mu$, Policy $\pi$, Discount factor $\gamma$, Precision parameter $\varepsilon$
Result: Vector $\boldsymbol{v}$ s.t. $v_s \approx V^\pi_\mu(s), \forall s \in \mathcal{S}$
$\boldsymbol{v} \leftarrow \mathbf{0}$
repeat
    $t \leftarrow 0$
    $s_0 \leftarrow$ starting state
    $\delta_{\mathtt{max}} \leftarrow 0$
    repeat
        $a_t \sim \pi(\cdot \mid s_t)$
        $s_{t+1} \sim \tau_a(s_t)$
        $\delta \leftarrow \rho(s_{t+1}) + \gamma v_{s_{t+1}} - v_{s_t}$
        $v_{s_t} \leftarrow v_{s_t} + \alpha\delta$
        $\delta_{\mathtt{max}} \leftarrow \max\{|\delta|, \delta_{\mathtt{max}}\}$
        $t \leftarrow t + 1$
    until $s_t$ is terminal;
until $\delta_{max} \leq \varepsilon$;
return $\boldsymbol{v}$
```

current belief over the state-action values $\boldsymbol{Q}^\pi$. The convergence properties to $\boldsymbol{Q}^*$ rely on the policy's ability to balance between exploration and exploitation [12], and most importantly, that π is gradually tuned to be more greedy w.r.t. $\boldsymbol{Q}^\pi$. The update is then done on the these state-action values:

$$Q^\pi_{s_t,a_t} \leftarrow Q^\pi_{s_t,a_t} + \alpha\big(\rho(s_{t+1}) + \gamma Q^\pi_{s_{t+1},a_{t+1}} - Q^\pi_{s_t,a_t}\big). \tag{2.19}$$

In Q-learning, the difference is subtle; the definition of an off-policy method is that it does not learn the value function of the policy followed. In this case, the value function learned is still Q^* but the only requirement on the policy is that all state-action pairs are visited and updated. The update is then given by:

$$Q_{s_t,a_t} \leftarrow Q_{s_t,a_t} + \alpha\big(\rho(s_{t+1}) + \gamma \max_a Q_{s_{t+1},a} - Q_{s_t,a_t}\big). \tag{2.20}$$

2.3.1 Least-squares methods

This section describes least-squares methods of approximating the value function of a policy. This is the same task as the previously mentioned algorithms are solving, but the solutions presented here are adapted to handle large state spaces.

Feature representation

From here on, the value functions will be approximated with feature representations of the state-action pairs, mainly so that large state spaces can be handled more efficiently.

For the state-action value function used in this section, the approximation is defined by the linear combination

$$\widehat{Q}(s,a) \triangleq \boldsymbol{\phi}(s,a)^\top \boldsymbol{w}, \qquad \boldsymbol{\phi}(s,a), \boldsymbol{w} \in \mathbb{R}^k, \tag{2.21}$$

or by the more compact notation

$$\widehat{\boldsymbol{q}} = \boldsymbol{\Phi}\boldsymbol{w}, \qquad \boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times k}, \boldsymbol{w} \in \mathbb{R}^k. \tag{2.22}$$

Where each feature vector, given by the function $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$, is comprised of k different, generally nonlinear, functions of the state-action pairs. The symbol $\boldsymbol{q}$ will be used to denote any value function, and $\widehat{\boldsymbol{q}}$ to denote a value function that lies in the subspace spanned by $\boldsymbol{\Phi}$.

Bellman residual minimisation

This section acts as a bridge to the explanation of the LSTDQ algorithm presented in the next section, and it partly follows the presentation made by its authors in [21]. It is a natural extension to solving the linear system for the value function when it is approximated by a linear combination of features.

Recall the Bellman equation for $Q^\pi(s,a)$ in Equation (2.9) and note that it has the linear system representation

$$\boldsymbol{q}^\pi = \boldsymbol{r} + \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{q}^\pi, \tag{2.23}$$

where $\boldsymbol{q}, \boldsymbol{r} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, $r_{(s,a)} = \mathbb{E}\big[\rho(s') \mid s,a\big]$ and $\boldsymbol{\mathcal{T}}^\pi \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}||\mathcal{A}|}$. In this context, $\boldsymbol{\mathcal{T}}^\pi$ is the policy induced transition kernel of a Markov chain with transitions of the kind $(s,a) \to (s',a')$ with probability $\mathcal{T}^\pi_{(s,a),(s',a')}$. The summation over pairs (s',a') in (2.9), where the expectation is given a state-action pair (s,a), can be seen as $\boldsymbol{\mathcal{T}}^\pi \boldsymbol{q} = \boldsymbol{\mathcal{T}}\boldsymbol{\Pi}_\pi \boldsymbol{q}^\pi$ where $\mathcal{T}_{(s,a),s'} = \tau_a(s' \mid s)$ and $\Pi_{s',(s',a')} = \pi(a' \mid s')$.

The right hand side of the linear system in (2.23) is often defined as the *Bellman operator* $\mathcal{L}_\pi$ applied to the left hand side. In other words, a shorthand for the linear system is $\boldsymbol{q}^\pi = \mathcal{L}_\pi \boldsymbol{q}^\pi$, and in an iterative policy evaluation scheme the updates are $\boldsymbol{q}_{n+1} \leftarrow \mathcal{L}_\pi \boldsymbol{q}_n$.

Replacing $\boldsymbol{q}^\pi$ with $\boldsymbol{\Phi}\boldsymbol{w}^\pi$ gives an overconstrained system (since $k < |\mathcal{S}||\mathcal{A}|$) with the least-squares solution:

$$\boldsymbol{w}^\pi = \Big((\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{\Phi})^\top (\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{\Phi})^\top \Big)^{-1} (\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{\Phi})^\top \boldsymbol{r} \tag{2.24}$$

This is a natural approach that mimics everything that we have learned so far; the only difference being that a linear feature approximation makes the system overconstrained. The interesting property to remember, however, is that the solution minimises the *Bellman residual*, i.e. the L_2 distance taken by $\mathcal{L}_\pi$ when applied to $\boldsymbol{\Phi}\boldsymbol{w}$. This helps in understanding why LSTDQ takes a different approach; and why that approach may be better.

Least-squares Temporal Difference Learning (LSTD$\boldsymbol{Q}$)

Note that the Bellman operator applied to $\boldsymbol{\Phi w}$, a point in the feature plane, gives a resulting point that is generally outside that plane. The *Bellman residual minimisation* above minimises this distance, but LSTDQ attempts to find a solution such that, when the Bellman operator is applied to it, the resulting vector is still in the feature plane:

$$\widehat{\boldsymbol{q}} \approx \mathcal{L}_\pi \widehat{\boldsymbol{q}}. \tag{2.25}$$

The equation can then be expressed as follows. The value function should be invariant to the combined operation of applying the Bellman operator and then projecting the result onto the feature plane:

$$\widehat{\boldsymbol{q}}^\pi = \boldsymbol{P}_{\boldsymbol{\Phi}} \mathcal{L}_\pi \widehat{\boldsymbol{q}}^\pi, \tag{2.26}$$

where the orthogonal projection $\boldsymbol{P}_{\boldsymbol{\Phi}}$ onto the subspace spanned by $\boldsymbol{\Phi}$ is given by $\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top$ (cf. [22] pp. 430 eq. (5.13.3)). The solution to this system is

$$\boldsymbol{w}^\pi = \Big(\boldsymbol{\Phi}^\top\big(\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{\Phi}\big)\Big)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{r}, \tag{2.27}$$

whose derivation can be found in [21] and is left out here for brevity. This method tries to find the point in the feature plane closest to the true value function. If the real value function is actually inside the feature plane, the two different methods give the same result.

The algorithm proceeds as follows. Given some data $\boldsymbol{D} = \big\{(s_i, a_i, s_i', r_i)\big\}_{i=1}^L$ of transitions, let $f : |\mathcal{S}||\mathcal{A}| \to \mathbb{R}$ be a probability distribution over the state-action pairs, s.t. $f_{\boldsymbol{D}}$ is the true distribution of $\boldsymbol{D}$, and let Δ_f be a diagonal matrix with elements $f(s,a)$. Then the LSTDQ algorithm builds empirical estimates of

$$\boldsymbol{A} = \boldsymbol{\Phi}^\top \Delta_{f_{\boldsymbol{D}}} \big(\boldsymbol{\Phi} - \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{\Phi}\big), \qquad \text{and} \tag{2.28}$$

$$\boldsymbol{b} = \boldsymbol{\Phi}^\top \Delta_{f_{\boldsymbol{D}}} \boldsymbol{r} \tag{2.29}$$

given by

$$\widetilde{\boldsymbol{A}} = \frac{1}{L}\sum_{i=1}^{L} \boldsymbol{\phi}(s_i,a_i)\big(\boldsymbol{\phi}(s_i,a_i) - \gamma \boldsymbol{\phi}(s_i', \pi(s_i'))\big)^\top, \qquad \text{and} \tag{2.30}$$

$$\widetilde{\boldsymbol{b}} = \frac{1}{L}\sum_{i=1}^{L} \boldsymbol{\phi}(s_i,a_i) r_i \tag{2.31}$$

s.t. the solution is given by $\widetilde{\boldsymbol{w}}^\pi = \widetilde{\boldsymbol{A}}^{-1}\widetilde{\boldsymbol{b}}$. In the limit of $L \to \infty$, this gives the true solution biased by the distribution of the data $f_{\boldsymbol{D}}$. LSTDQ is summarised in Algorithm 3.

Algorithm 3: LSTDQ

```
Input: Data $\boldsymbol{D}$, Basis functions $\boldsymbol{\phi}$, Discount factor $\gamma$, Policy $\pi$
Result: Vector $\boldsymbol{w}$ s.t. $Q^{\pi}(s,a) \approx \boldsymbol{\phi}(s,a)^{\top}\boldsymbol{w}$
$\widetilde{\boldsymbol{A}} \leftarrow \boldsymbol{0}$
$\widetilde{\boldsymbol{b}} \leftarrow \boldsymbol{0}$
foreach $(s,a,s',r) \in \boldsymbol{D}$ do
    $\widetilde{\boldsymbol{A}} \leftarrow \widetilde{\boldsymbol{A}} + \boldsymbol{\phi}(s,a)\big(\boldsymbol{\phi}(s,a) - \gamma\boldsymbol{\phi}(s',\pi(s'))\big)^{\top}$
    $\widetilde{\boldsymbol{b}} \leftarrow \widetilde{\boldsymbol{b}} + \boldsymbol{\phi}(s,a)r$
end
return $\widetilde{\boldsymbol{w}}^{\pi} = \widetilde{\boldsymbol{A}}^{-1}\widetilde{\boldsymbol{b}}$
```

Least-squares Policy Iteration (LSPI)

LSTDQ is an extension of LSTD from previous work in [23, 24] which is based on state values V. The issue with LSTD, as argued by [21], is that it cannot be used for action selection without a model of the underlying process. For instance, had the model $\mathcal{T} \in \mu$ been given, together with a value function V a policy π with the property $V \equiv V^{\pi}$ can always be constructed as $\pi(s) = \arg\max_a\big\{\sum_{s'\in\mathcal{S}} \tau_a(s' \mid s)\big(\rho(s') + \gamma V(s')\big)\big\}$. This concern is relevant in board games, where the actions made by the opponent are an unknown stochastic part of the model.

When state-action values Q are given, the policy can be constructed directly by $\pi(s) = \arg\max_a\big\{Q(s,a)\big\}$. To demonstrate the benefits, the authors also suggest a model-free policy iteration algorithm for finding an optimal[4] policy by using LSTDQ. The algorithm is named *Least Squares Policy Iteration* (LSPI) and is presented in Algorithm 4.

Algorithm 4: LSPI

```
Input: Data $\boldsymbol{D}$, Basis functions $\boldsymbol{\phi}$, Discount factor $\gamma$, Precision parameter $\varepsilon$,
       Initial policy $\pi_0$
Result: Policy $\pi$
$\pi' \leftarrow \pi_0$
repeat
    $\pi \leftarrow \pi'$
    $\pi' \leftarrow \texttt{LSTD}Q(\boldsymbol{D},\boldsymbol{\phi},\gamma,\boldsymbol{w})$
until $\|\boldsymbol{w}^{\pi} - \boldsymbol{w}^{\pi'}\| < \varepsilon$;
return $\pi$
```

[4]The performance of the policy π returned by LSPI is bounded by $\|\widehat{Q}^{\pi} - Q^{*}\|_{\infty} \leq \frac{2\gamma\varepsilon}{(1-\gamma)^2}$, details given in [21].

2.4 Learning from demonstrations

A value function roughly states *how* to accomplish a task, since it directly defines a deterministic policy. For example, in large and complex environments the goal may be to reach a specific terminal state, such as escaping a maze, retrieving an item or winning a board game. The value function then measures the proximity of such a goal. Stating a value function manually is thus a difficult task subject to bias in the resulting policy. Algorithms learning the value function directly also need to worry about generalising well to unseen states.

The reward function is a natural abstraction that instead states *what* to accomplish, e.g. by assigning a numerical reward of +1 to goal states and 0 to all others. This allows the algorithms discussed so far to find optimal value functions and optimal policies with no bias. However, for long and complex tasks in large environments, they become inefficient unless a proper set of *subgoals* can be defined. If certain subgoals do indeed signify progress towards the end goal, they could be assigned positive rewards; but identifying such subtasks is a difficult problem.

Inverse reinforcement learning [25, 26] is concerned with inferring the inherent preferences of a demonstrator or *expert* from its observed interactions with an environment. Constructing good reward functions thus becomes an inference task. This section explains IRL techniques for construction of reward functions through the use of *unlabeled data*, which is the common approach since data where the goals have been labeled is much less common.

When multiple experts act in the same environment, their individual subgoals may be different and the method explained here infer them in a *multiple task* setting. The main focus is on the multitask Bayesian approach proposed in [7]; an extension of its single task counterpart in [1]. This is reviewed in the next section. Some previous multitask Bayesian approaches include [27, 28, 29] but are not covered here.

2.4.1 Bayesian multitask inverse reinforcement learning

When dealing with IRL it is instructive to separate the MDP tuple into the parts $\mu = (\nu, \rho, \gamma)$, where $\nu = (\mathcal{S},\mathcal{A},\mathcal{T})$ is a *controlled Markov process* (CMP) defining the dynamics of the game/system. In this framework, we will distinguish between M different *tasks* $\{\mu_m = (\nu, \rho_m, \gamma)\}_{m=1}^{M}$ so that the only varying element between tasks is the reward function. For each task m, there is a set of demonstrations $\boldsymbol{d}_m$, s.t. the complete dataset is denoted by $\boldsymbol{D} = \{\boldsymbol{d}_m\}_{m=1}^{M}$. The corresponding sets of (unknown) reward functions and policies are denoted by $\boldsymbol{\rho} = \{\rho_m\}_{m=1}^{M}$ and $\boldsymbol{\pi} = \{\pi_m\}_{m=1}^{M}$ respectively.

The following model employs a hierarchical Bayesian approach with a hyperprior η on the joint reward-policy priors ϕ. Since η is defined on the joint reward-policy *distribution function* space, rather than on the parameters of some fixed functions' parameters, it becomes the only parameter to the model in a very general sense.

As the first of two examples demonstrated in [1], if η is chosen as the distribution on a product prior ϕ on *reward functions* and *policy Softmax temperatures* (i.e. noise levels, see Section 3.3), it then jointly determines unique policies for which the likelihood of the data is well defined. This is clear because a sampled reward function ρ gives rise to an optimal value function Q^*_ρ which induces a policy π that follows Q^*_ρ according to a well defined probability with noise. From here on, the notation $Q_\rho \triangleq Q_\mu$, where $\mu = (\nu,\rho,\gamma)$, will be used to emphasis the dependency on ρ.

The second formulation, to be used here, lets η be a distribution on priors ϕ on *policy functions* and on the *optimality* of the policies, leading to an implicit distribution on reward functions conditional on policies. The posterior probability of a reward function can then be calculated as the marginal over a set of sampled posterior policies. This is the heart of the algorithm, to be made more clear here.

Model

The joint reward prior ψ and policy prior ξ, denoted by ϕ, is sampled from the hyperprior η. The notation ϕ will not be used very much, but it is connected to the individual priors in the sense that ϕ is a probability measure on the reward-policy product space $\mathcal{R} \times \mathcal{P}$ according to $\phi(R,P \mid \nu) \triangleq \int_R \xi(P \mid \nu) \mathrm{d}\psi(\rho \mid \nu)$ for $R \subset \mathcal{R}$ and $P \subset \mathcal{P}$. $\psi(\rho_m = \rho \mid \nu)$ denotes the prior probability that the m:th reward function is ρ, and $\xi(\pi_m = \pi \mid \nu)$ denotes the prior probability that the m:th policy is π. The dependencies on ν will be left out for clarity, since it stays constant throughout the different tasks. The corresponding posteriors are denoted by $\psi(\cdot \mid \boldsymbol{D})$ and $\xi(\cdot \mid \boldsymbol{D})$. The model is shown in Figure 2.2.

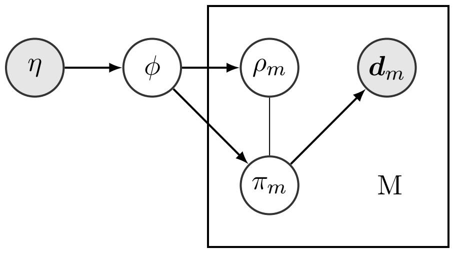


Figure 2.2: Graphical model of the reward-policy priors. Darker color indicates observables. Here, ρ_m,π_m is sampled jointly from ϕ, indicated by the undirected edge between them. The data $\boldsymbol{d}_m$ is generated by the m:th demonstrator's policy π_m.

Derivation

The purpose of Bayesian IRL is to calculate the posterior probability $\psi(\cdot \mid \boldsymbol{D})$ on reward functions. As mentioned, this model does so by approximating the marginal over policy posteriors

$$\psi(\rho \mid \boldsymbol{D}) = \sum_{\pi \in \mathcal{P}} \psi(\rho \mid \pi)\xi(\pi \mid \boldsymbol{D}) \tag{2.32}$$

by sampling from $\xi(\cdot \mid \boldsymbol{D})$. The policy posterior is simple to calculate if, for instance, $\xi \sim \mathrm{Dir}(\alpha)$ is chosen. Thus, the main goal is to find a way to express and approximate $\psi(\rho \mid \pi)$.

Let the loss of policy π w.r.t. reward function ρ be defined as

$$\ell_\rho(\pi) \triangleq \max_s \left\| V_\rho^*(s) - V_\rho^\pi(s) \right\|. \tag{2.33}$$

A policy is said to be ε-*optimal* w.r.t. ρ if $\ell_\rho(\pi) < \varepsilon$. Let the prior probability that $\ell_\rho(\pi) = \varepsilon$ be given by $\beta(\varepsilon \mid \pi)$ for any ρ, and assume $\beta(\varepsilon \mid \pi) = \beta(\varepsilon)$[5]. Then $\beta([0,\varepsilon])$ is the prior probability that the policy is ε-optimal. This opens up the possibility to calculate $\psi(\rho \mid \pi)$ via

$$\psi(\rho \mid \pi) = \int_0^\infty \psi(\rho \mid \varepsilon,\pi) \mathrm{d}\beta(\varepsilon), \tag{2.34}$$

where $\psi(\rho \mid \varepsilon,\pi)$ can be interpreted as a prior probability measure on $\mathcal{R}$ given that π is ε-optimal. The marginal posterior (2.32) can then be written as

$$\psi(\rho \mid \boldsymbol{D}) = \sum_{\pi \in \mathcal{P}} \left(\int_0^\infty \psi(\rho \mid \varepsilon,\pi) \mathrm{d}\beta(\varepsilon) \right) \xi(\pi \mid \boldsymbol{D}). \tag{2.35}$$

The next task is then to find a motivated expression for $\psi(\rho \mid \varepsilon,\pi)$. Let $\mathcal{R}_\varepsilon^\pi \triangleq \{\rho \in \mathcal{R} : \ell_\rho(\pi) < \varepsilon\}$ be the set of reward functions for which π is ε-optimal. Then the following definition is motivated in [1] by using a counting measure on $\mathcal{R}$:

$$\psi(\rho \mid \varepsilon,\pi) \triangleq \frac{\mathbb{1}_{\mathcal{R}_\varepsilon^\pi}(\rho)}{|\mathcal{R}_\varepsilon^\pi|}, \tag{2.36}$$

which can be seen as an unnormalised prior on reward functions.

Assuming that some finite set $\mathcal{R}$ is given, and that K policies $\pi^{(1)},\dots,\pi^{(k)},\dots,\pi^{(K)} \overset{iid}{\sim} \xi(\cdot \mid \boldsymbol{D})$ are sampled from the posterior, a loss matrix $\boldsymbol{L} \in \mathbb{R}^{K \times |\mathcal{R}|}$ can be constructed from (2.33). One final observation remains for (2.32) to be approximated. Let $(\varepsilon_i)_{i=1}^{K \times |\mathcal{R}|}$ be a monotonically increasing sequence of the elements of $\boldsymbol{L}$. Then $\psi(\rho \mid \varepsilon,\pi) = \psi(\rho \mid \varepsilon',\pi)$ for any $\varepsilon,\varepsilon' \in [\varepsilon_i,\varepsilon_{i+1}]$, and the approximation is finally given by:

$$\begin{aligned} \widehat{\psi}(\rho \mid \boldsymbol{D}) &= \sum_{k=1}^{K} \sum_{i=1}^{K|\mathcal{R}|} \psi(\rho \mid \epsilon_i,\pi^{(k)}) \beta([\epsilon_i,\epsilon_{i+1}]) && (2.37) \\ &= \sum_{k=1}^{K} \sum_{i=1}^{K|\mathcal{R}|} \frac{\mathbb{1}_{\mathcal{R}_{\varepsilon_i}^{\pi^{(k)}}}(\rho)}{|\mathcal{R}_{\varepsilon_i}^{\pi^{(k)}}|} \beta([\epsilon_i,\epsilon_{i+1}]). && (2.38) \end{aligned}$$

Algorithm

The end result is a Monte Carlo approximation scheme shown in Algorithm 5; K being the number of MC samples. Here, the hyperprior generates ξ (i.e. α if $\xi \sim \mathrm{Dir}(\alpha)$) and a parameter for the chosen form of the optimality prior β.

[5]This is not to be confused with the Beta distribution. In fact, later, β is chosen to be the exponential distribution.

Algorithm 5: Bayesian Multitask IRL - Monte Carlo approximation

```
Input: Reward functions $\mathcal{R}$
for $k = 1, \ldots, K$ do
    $(\xi^{(k)}, \beta^{(k)}) \sim \eta$
    for $m = 1, \ldots, M$ do
        $\pi_m^{(k)} \sim \xi(\cdot \mid \boldsymbol{d}_m)$
    end
end
Calculate $\widehat{\psi}(\cdot \mid \boldsymbol{d}_m)$ from (2.37) and $\left\{\pi_m^{(k)}\right\}_{k=1}^{K}$ for all $m$
```

3

Model

This chapter explains the choices required to implement the Bayesian multitask Inverse Reinforcement Learning algorithm (BMTIRL) (Algorithm 5).

Since the algorithm calculates a probability distribution on the reward space $\mathcal{R}$, the expected reward function

$$\widehat{\rho}_m \triangleq \sum_{\rho \in \mathcal{R}} \psi(\rho \mid \boldsymbol{d}_m)\rho \tag{3.1}$$

and its corresponding optimal policy $\widehat{\pi}^*_m \triangleq \pi^*_{\widehat{\rho}_m}$, where $\pi^*_\rho \triangleq \pi^*_{\mu=(\nu,\rho,\gamma)}$, will be used to denote the outputs of the algorithm. The *true* m:th reward function denoted by ρ_m represents the m:th expert's intrinsic goals, and technically means that the agent believes that it is interacting with the environment $\mu_m = (\nu,\rho_m,\gamma)$ in which it is not necessarily optimal. All experts are however acting in the same *real* environment $\mu = (\nu,\rho,\gamma)$ and it is assumed that each ρ_m is drawn from some distribution with mean ρ. To approximate ρ, the combined reward function across all experts will be used and is denoted by

$$\widehat{\rho} \triangleq \frac{1}{M}\sum_{m=1}^{M} \widehat{\rho}_m, \tag{3.2}$$

with the corresponding optimal policy denoted by $\widehat{\pi}^* \triangleq \pi^*_{\widehat{\rho}}$.

Section 3.5 discusses how to construct $\mathcal{R}$ when there exists some approximation of the optimal value function. The hope is that the value $V^*_{\widehat{\rho}}$ of the optimal policy inferred from all experts is somehow closer to the true optimal value function V^*_ρ than any (or most) of the individual value functions $V^*_{\widehat{\rho}_m}$. This may be due to the fact that different experts contribute to different parts of the feature vector that is used to approximate V. Figure 3.1 shows a schematic representation of this assumption.

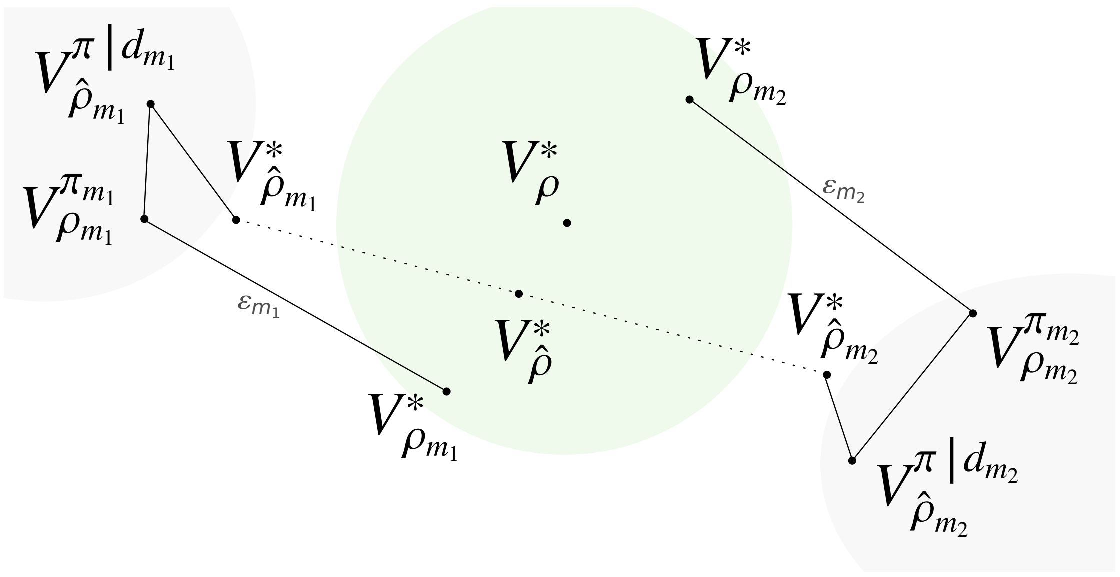

$V^{\pi\,|\,d_{m_1}}_{\hat{\rho}_{m_1}}$
$V^{\pi_{m_1}}_{\rho_{m_1}}$
$V^{*}_{\hat{\rho}_{m_1}}$
V^{*}_{ρ}
$V^{*}_{\rho_{m_2}}$
ε_{m_2}
ε_{m_1}
$V^{*}_{\hat{\rho}}$
$V^{*}_{\hat{\rho}_{m_2}}$
$V^{\pi_{m_2}}_{\rho_{m_2}}$
$V^{*}_{\rho_{m_1}}$
$V^{\pi\,|\,d_{m_2}}_{\hat{\rho}_{m_2}}$

Figure 3.1: A schematic representation of the assumed relationship between the value functions for the different environments considered in the inference process. In the center of the figure lies the true value function V^*_ρ of optimal play, and close to it are two different experts' intended value functions $V_{\rho_{m_{1,2}}}$. Implied by the colored areas around the experts' actual values $V^{\pi_{m_{1,2}}}_{\rho_{m_{1,2}}}$ are all possible value functions from pairs of sampled policies and reward functions. The approximated optimal value $V^*_{\widehat{\rho}}$ derived from $\widehat{\rho}_{m_{1,2}}$ is assumed to lie close to the true optimal value V^*_ρ.

3.1 Features

Although features for state-action pairs have been mentioned, a definition has not been chosen yet. In many board games an action implies a local interaction within a subregion of the board. It is thus a natural choice for a state-action feature vector to be composed of features from this region only. However, for simplicity, the choice made here is to define the state-action feature vector as the expected vector given by transition probabilities for action a in state s:

$$\boldsymbol{\phi}(s,a) \triangleq \sum_{s' \in \mathcal{S}} \tau_a(s' \mid s)\boldsymbol{\phi}(s), \tag{3.3}$$

where $\boldsymbol{\phi}(s)$ is a set of nonlinear functions of the state s independent of the previous action. This method of averaging feature vectors has some convenient properties, as shall be seen later. The chosen features for the different environments are explained in Section 3.2.

3.1.1 Reward function feature representation

Recall that the rewards are chosen as a function of the state transitioned *to*, although in the most general case they may be a function of all parameters from the transition (such as the action). Regarding the argument made in Section 3.1 above, this would have allowed for the combination of state-action features and rewards defined on them. For the applications discussed here, however, it will be sufficient to continue with the current convention and to choose the definition:

$$\rho_{\boldsymbol{w}}(s) \triangleq \boldsymbol{\phi}(s)^\top \boldsymbol{w}. \tag{3.4}$$

3.2 Environments

This section explains the MDP environments that will be examined.

3.2.1 Random MDP

The feature vector of a state in the Random MDP state space is chosen to have the same length as the feature space $|\mathcal{S}|$ and to have the following *one-hot encoding* representation:

$$\phi_i(s) \triangleq \begin{cases} 1 \text{ if } i = s \\ 0 \text{ otherwise} \end{cases} \tag{3.5}$$

Note that, together with (3.3), this definition leads to the property that $\phi_{s'}(s,a) \equiv \tau_a(s' \mid s)$, i.e. that the state-action feature vector is a list of transition probabilities for the given action.

The transition kernel $\mathcal{T}$ will be sampled so that the $\tau \in \mathcal{T}$ are given by $\boldsymbol{\tau}_a(\cdot, s) \overset{iid}{\sim} \mathrm{Dir}(\alpha)$ for all $(a,s) \in \mathcal{A} \times \mathcal{S}$.

The reward function is sampled according to Equation (3.4) and $\boldsymbol{w} \sim \mathrm{Dir}(\alpha)$. Note that, together with (3.3) and (3.5), this implies that $\rho_{\boldsymbol{w}}(s) \equiv w_s$.

3.2.2 Tic-tac-toe

Tic-tac-toe is the famous game where two players, referred to as `X` and `O`, take turns placing symbols on a board of size 3×3 until either one player has three symbols in a row (in any orientation) or the board is filled without a winner (a tie).

Tic-tac-toe is chosen for its state space being large enough for approximation by value iteration to be impractical, but small enough for having a known optimal policy and natural features. Although the state space can in fact be reduced to manageable sizes by considering equivalent reflections and rotations of the board, this is not the purpose of the work and will not be done here. It will be handled as though the state space is indeed of size $3^9 = 19683$, i.e. by representing the state by an integer in the range [0,19683], although most of those states are not even legal positions. The number of legal actions varies between 0 to 9 depending on the number of occupied locations of the board (of size 3×3).

To make the translation to an MDP simpler, the dynamics will have player `X` in focus, so that every transition leads to a state where it is `X`'s turn to play unless the game ends.

The opponent is the random player, so that the transition probabilities given a move by `X` are uniformly distributed by the available remaining moves for `O`.

An optimal strategy is given by a heuristic set of rules in [30], which will be referred to as the *programmatic optimal policy.*[1]

Features

The feature vector for a state in the Tic-tac-toe environment is chosen to be a subset of the features discussed in [31], namely, for *each player* `X` and `O`: the number of *singlets* (lines in any orientation with exactly one symbol), *doublets* (like singlets but exactly *two* identical symbols), *triplets* (three identical symbols in a row), *crosspoints* (the number of empty points that belong to at least two singlets), *corners* (the number of occupied corners), and finally a center occupation feature with value in $\{-1,0,1\}$ for `O`, `empty` and `X` respectively. Apart from these features, an extra *fork* feature is added, defined as the number of *distinct* doublets - 1. Apart from the center occupation feature, all features are defined for both players, which makes a total of $k = 13$ features.

The choice of adding the fork feature was made so that the programmatic optimal policy could be implemented by the use of features and one step lookahead alone.

Rewards

True rewards are given in the terminal states (win or tie) as $\{-1,0,1\}$ for a loss, tie and win respectively. Let x be the index of the triplets feature for `X` and o the corresponding index for `O`, then the *true* reward function $\rho_{\boldsymbol{w}}$ is defined by

$$w_i = \begin{cases} 1 \text{ if } i = x, \\ -1 \text{ if } i = o, \\ 0 \text{ otherwise.} \end{cases} \tag{3.6}$$

Naturally, when a reward function $\rho \in \mathcal{R}$ during inference in BMTIRL is considered, it will be used instead of (3.6).

3.3 Generation of demonstrations

Recall that an optimal policy given a value function $Q(s,a)$ is based on the maximisation $\pi_Q^*(s) \triangleq \arg\max_a \{Q(s,a)\}$. A common alternative [13] that makes the policy (infinitely) differentiable is to define a *Softmax* policy according to

$$\pi_Q^{\mathit{Softmax(c)}}(a \mid s) \triangleq \frac{\mathrm{e}^{Q(s,a)/c}}{\sum_{a' \in \mathcal{A}} \mathrm{e}^{Q(s,a')/c}}, \tag{3.7}$$

[1]As is also referenced and reviewed in `http://en.wikipedia.org/wiki/Tic-tac-toe#Strategy`; accessed at 2014-05-10.

where c is a temperature parameter with the properties $\lim_{c \to 0} \pi_Q^{\mathit{Softmax}(c)} = \pi_Q^*$ and that $\lim_{c \to \infty} \pi_Q^{\mathit{Softmax}(c)}$ is the uniformly random policy. It will be used here to generate demonstrations so that the noise level can be controlled. No real data from human play will be used.

Each expert m generates a dataset $\boldsymbol{d}_m$ (see Figure 2.2) consisting of N independent trajectories of total lengths $T_1, \ldots, T_n, \ldots, T_N$:

$$\boldsymbol{d}_m = \left\{ \left((s_t^{(n)}, a_t^{(n)}) \right)_{t=1}^{T_n} \right\}_{n=1}^{N}. \tag{3.8}$$

Each trajectory n starts in a state s_0 drawn from some initial state distribution, actions are sampled by $a_t \sim \pi_Q^{\mathit{Softmax(c)}}(\cdot \mid s_t)$ and states by $s_{t+1} \sim \tau_{a_t}(\cdot \mid s_t)$ until $s_t = s_{T_n}$. The termination may be chosen either by setting a fixed demonstration length or by just following the policy until a terminal state is reached[2].

3.4 Policy space

The approximation of the posterior reward distribution in Equation (2.37) relies on samples $\pi_m^{(k)}$ from $\pi_m \mid \boldsymbol{d}_m$, where the policy prior is chosen s.t. $\pi_m(\cdot \mid s) \sim \mathrm{Dir}(\alpha)$ for all $s \in \mathcal{S}$.

For a state s, the updates of the Dirichlet parameters $\alpha_a^{(s)}$ for each action a is given by simply adding the count of the number of times that (s,a) was observed in $\boldsymbol{d}_m$:

$$\alpha_a^{(s)} = \alpha_a^{(s)} + \sum_{n=1}^{N} \sum_{t=1}^{T_n} \mathbb{1}\left\{ s_t^{(n)} = s \;\wedge\; a_t^{(n)} = a : (s_t^{(n)}, a_t^{(n)}) \in \boldsymbol{d}_m \right\}, \tag{3.9}$$

which applies for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. At a first glance, this requires the enumeration of all parameters $\alpha_a^{(s)}$, which would be intractable since this work focuses on too large state spaces. However, it is still possible for a *lazy*[3] representation in the following sense. For every expert m, the posterior is stored as counts of observed state-action pairs in $\boldsymbol{d}_m$, and for every sampled policy $\pi_m^{(k)}$, a multinomial is sampled and stored only once for every s upon the querying of $\pi_m^{(k)}(\cdot \mid s)$.

3.4.1 Loss calculation

The loss calculation $\ell_\rho(\pi)$ in (2.33) amounts to estimating V_ρ^π and V_ρ^* for all states, but since all states cannot be enumerated, a subset $S \subset \mathcal{S}$ has to be used instead. Also, due to that the posterior policy will have updated parameters only in *observed states*, this motivates the choice of using $S = \{s : s \in \boldsymbol{d}_m\}$ for each expert m.

[2]This assumes that policy is *proper* and is guaranteed to reach a terminal state, which holds trivially for the MDPs focused on here.

[3]Lazy initialisation — An object is not calculated before it is requested.

To actually estimate V_ρ^π, LSTDQ (Algorithm 3) is used to get a vector of weights $\boldsymbol{w}$ for which $Q_\rho^\pi \approx \boldsymbol{\phi}(s,a)^\top \boldsymbol{w}$. Similarly, LSPI (Algorithm 4) is used to retrieve π_ρ^* and Q_ρ^*. As LSTDQ relies on sampled transitions from some distribution (where the only requirement is that the distribution visits all states), a completely random policy will be used to generate the LSTDQ-specific demonstrations for the random MDP.

3.5 Reward function space

This section connects the assumed value function space in Figure 3.1 with the generation of the reward space $\mathcal{R}$. If an approximation of the optimal value function exists, the experts' intended value functions should also be close. Using a Gaussian approximation on state-action values, the distribution on the reward space can be solved for and the discrete reward space $\mathcal{R}$ can be sampled.

In Equation (2.23), the linear system $\boldsymbol{q}^\pi = \boldsymbol{r} + \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{q}^\pi$ was based on Equation (2.9) where $r_{(s,a)} = \mathbb{E}[\rho(s') \mid s,a]$. Using definitions (3.4) and (3.3) for the reward function and state-action feature function respectively results in the equivalences $r_{(s,a)} \triangleq \mathbb{E}[\rho(s') \mid s,a] \equiv \boldsymbol{\phi}(s,a)^\top \boldsymbol{w}$ and $\boldsymbol{r} \equiv \boldsymbol{\Phi}\boldsymbol{w}$, where $\boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times k}$ as in Section 2.3.1.[4]

Let $\boldsymbol{r} = \boldsymbol{q} - \gamma \boldsymbol{\mathcal{T}}^\pi \boldsymbol{q}$ denote the target values in a linear system $\boldsymbol{r} = \boldsymbol{\Phi}\boldsymbol{w}$, so that its least-squares soluton is $\boldsymbol{w} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi} \boldsymbol{r}$. Now assume that there exists some multivariate normal approximation $\widetilde{\boldsymbol{q}}$ of $\boldsymbol{q}$ with known correlations. With assumpions on the game dynamics, this induces a transformed distribution for $\widetilde{\boldsymbol{r}}$ denoted by $\boldsymbol{\mu_r}, \boldsymbol{\Sigma_r}$.

This is different from regular Bayesian linear regression since, there, it is assumed that every single target value r_i has some unknown univariate i.i.d. noise $\varepsilon_i \sim \mathcal{N}(0,\sigma^2)$ added to it. In the above context, $(\boldsymbol{\mu_r},\boldsymbol{\Sigma_r})$ is already given and the problem is to calculate $(\boldsymbol{\mu_w},\boldsymbol{\Sigma_w})$. The solution is calculated in [32] where the following expressions are given by (14.11) and (14.12) in the cited work:

$$\boldsymbol{\mu_w} = (\boldsymbol{\Phi}^\top \boldsymbol{\Sigma_r}^{-1} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\Sigma_r}^{-1} \boldsymbol{\mu_r}, \tag{3.10}$$

$$\boldsymbol{\Sigma_w} = (\boldsymbol{\Phi}^\top \boldsymbol{\Sigma_r}^{-1} \boldsymbol{\Phi})^{-1}. \tag{3.11}$$

Each reward function $\rho_i \in \mathcal{R}$, where $\rho_i(s) = \boldsymbol{\phi}(s)^\top \boldsymbol{w}_i$, is thus sampled according to:

$$\boldsymbol{w}_i \overset{iid}{\sim} \mathcal{N}(\boldsymbol{\mu_w},\boldsymbol{\Sigma_w}), \tag{3.12}$$

which is equivalent to the slower procedure of — for every sample i — first sampling a set of values $\boldsymbol{q} \in \mathbb{R}^{|S||\mathcal{A}|}$ for some $S \subset \mathcal{S}$, and then solving the linear system mentioned above.

One limitation of BMTIRL is that it has to operate on a finite reward space $\mathcal{R}$, which makes generating it an important task. The above procedure motivates why it can be sampled using random playouts. In fact, in a message passing framework employed in

[4]Take careful note, however, that the weight vector $\boldsymbol{w}$ used in $\rho_{\boldsymbol{w}}$ is not the same as the one used in Section 2.3.1 for value functions. Throughout this chapter, $\boldsymbol{w}$ is used only to describe the reward function, hence a minimal risk for confusion.

[33], they motivate that only *one random playout* is necessary under the assumption that terminal states with similar outcomes are clustered. The same approach will be used here to build estimates of the values.

4

Experiments

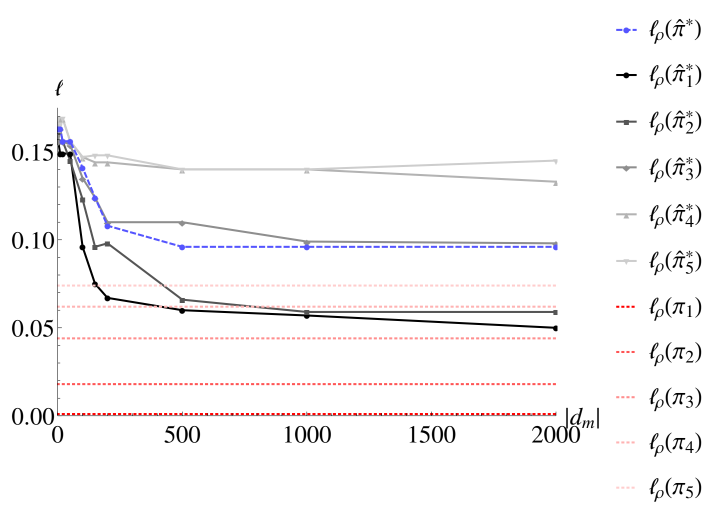
The general setup of the experiments are as follows.

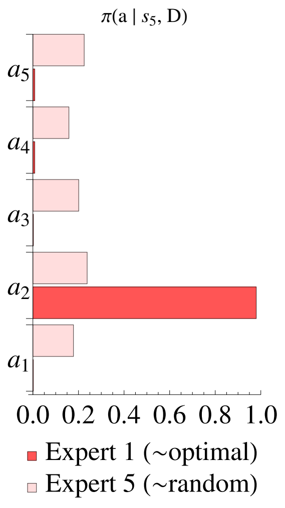
Given an increasing amount of data from each expert, the quality of the inference was examined. In particular, the model (3.1) assumes that each expert m may have a different distribution on reward functions, and it does not know whether the experts act in the same or similar environments (same or different ρ_m). Thus, convergence is shown for all experiments by measuring the loss $\ell_{\rho_m}(\widehat{\pi}^*_m)$ of the m:th inferred optimal policy (that is optimal w.r.t. the estimated reward function for each individual expert $\widehat{\rho}_m$), evaluated in the m:th true environment given by ρ_m. When all experts use the same reward function, this simplifies somewhat to the *single task* setting where the true reward function $\rho_m = \rho$ will be chosen for all experts. The convergence measure is then (equivalently) $\ell_{\rho}(\widehat{\pi}^*_m)$.

The *performance* is shown by measuring the loss $\ell_{\rho}(\widehat{\pi}^*)$ of the policy $\widehat{\pi}^*$ (that is optimal w.r.t. the inferred expected reward function $\widehat{\rho}$ from (3.2)), evaluated in the true environment given by ρ. Note the similarity to the *convergence* measure in the previous paragraph, and that in the single task setting the convergence and performance are measured in the same space and can be viewed together, whereas in the multitask setting these spaces are different and are viewed side by side.

4.1 Random MDP

The first experiment was to evaluate the algorithm a small randomly constructed MDP with 20 states and 5 actions available in each state. Each transition probability vector was sampled according to $\tau_a(\cdot \mid s) \overset{iid}{\sim} \mathrm{Dir}(\alpha = 1.0), \forall a \in \mathcal{A}$, and the true reward function $\rho_{\boldsymbol{w}}$ was sampled by $\boldsymbol{w} \sim \mathrm{Dir}(\alpha = 1.0)$. The set of proposal reward functions consisted of $|\mathcal{R}| = 20$ i.i.d. samples from the same distribution as the true reward function. The optimality prior β was set to the exponential distribution with parameter $\lambda = 10.0$, which roughly sets a cumulative probability to a loss < 0.05 to approximately 40%. A





(a) Losses of policies derived from inferred reward functions, where the policy derived from the average reward function is shown in dashed, blue.

(b) The posterior action probability for two different experts in state s_5.

Figure 4.1: Single task convergence **(a)** of random MDP with 20 states and 5 actions, 5 experts, $\gamma = 0.90$, $K = 30$ and $|\mathcal{R}| = 20$. The choice of parameters makes the first expert nearly optimal and the last expert close to random as shown in in **(b)**, due to Softmax action probabilities $\propto e^{Q(s,a)/c}$ depending implicitly on the MDP parameters via the magnitudes of the value function.

discounting of $\gamma = 0.9$ was used.

The first experiment examined the single task setting ($\rho_m = \rho, \forall m$) for 5 different experts by setting the m:th expert's Softmax temperature to $c_m = \{0.001, 0.005, 0.01, 0.015, 0.02\}_m$. The experiment was repeated many times, and the results presented in Figure 4.1 were representative. Increasing the size of the reward function space by $20 \rightarrow 60$ gave improved results shown in Figure 4.2. In another experiment not reported here, the number was increased even further to 120, without any difference in results.

The multitask setting and the hypothesised relationship between value functions (Figure 3.1) were examined by perturbing the weight vector of the reward function of each expert by adding i.i.d. Gaussian noise with variance $\sigma^2 = 4.0$ to each element, i.e. $\rho = \rho_{\boldsymbol{w}}, \rho_m = \rho_{\boldsymbol{w}_m}$ and $\boldsymbol{w}_m \sim \mathcal{N}(\boldsymbol{w}, 4.0)$. This value was chosen so that the performance $\ell_\rho(\pi_m)$ of the experts' policies π_m (which are optimal w.r.t. ρ_m) would be signifinantly different than those of π_ρ^*. The results presented in Figure 4.3 show that it is now possible to achieve a higher performance (lower loss) in comparison to those of the individual experts.

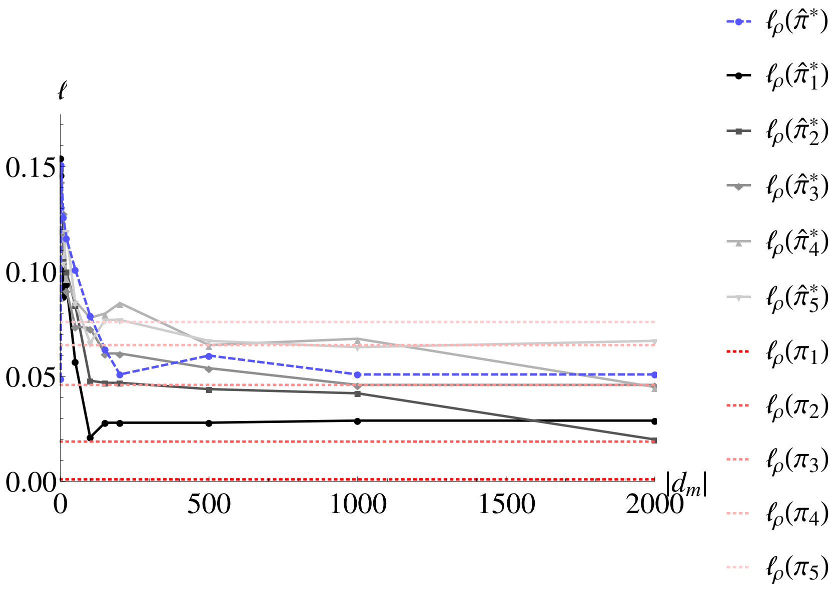
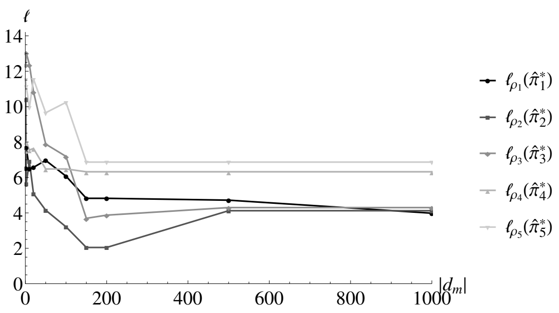
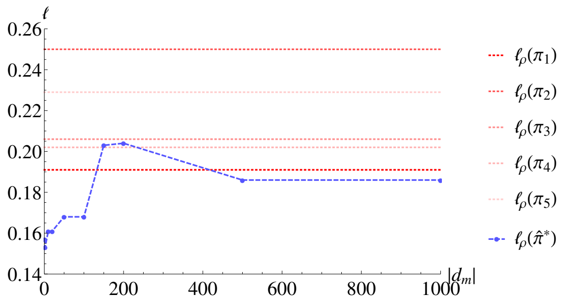


Figure 4.2: Random MDP with 20 states and 5 actions, 5 experts, $\gamma = 0.90$. The number of reward functions is increased to $|\mathcal{R}| = 60$, leading to lower losses reached.



(a) Multitask convergence of losses.



(b) Multitask optimal policy $\widehat{\pi}^*$ performance in true MDP compared to each expert.

Figure 4.3: Inference in the multitask setting where each expert has its own reward function $\rho_{\boldsymbol{w}_m}$ where $\boldsymbol{w}_m \sim \mathcal{N}(\boldsymbol{w},4.0)$ and $\rho_{\boldsymbol{w}}$ is the true reward function. Parameters are $K = 10, |\mathcal{R}| = 20, \gamma = 0.90$ and all expert temperatures are set to 0.001.

Feature	Prog.	LSPI
Singlets `X`	0.151	0.190
Doublets `X`	0.212	0.306
Triplets `X`	0.879	0.884
Crosspoints `X`	0.004	0.009
Corners `X`	0.072	0.068
Forks `X`	0.153	0.280
Singlets `O`	0.186	0.370
Doublets `O`	-0.063	-0.040
Triplets `O`	-1.352	-1.518
Crosspoints `O`	-0.258	-0.428
Corners `O`	0.100	0.099
Forks `O`	-0.921	-1.434
Center occupation	0.030	0.013

Table 4.1: Weights found by evaluating the programmatic optimal policy (*Prog.*) using LSTDQ, and those found by letting LSPI converge to an optimal policy from initial weights $\boldsymbol{w}_0 = \mathbf{0}$.

4.2 Tic-tac-toe

The second experiment involved evaluating the multitask setting of the algorithm in the Tic-tac-toe domain. First it was asserted that the programmatic optimal policy and the policy returned by LSPI had similar weights in their resulting value function. A number of 5000 random playouts were generated as data for LSTDQ. This gave the weights presented in Table 4.1. Some discrepancy does exist, but no further analysis was made as to why. Both policies have roughly the same win rate against the random policy, but their strategies may be different.

Each reward function was sampled by first sampling a set of state-action values and then solving the linear system given by those values, as explained in Section 3.5. The state-action values were sampled for every $(s,a) \in \boldsymbol{X}$ where $\boldsymbol{X}$ was the demonstrations from 10 completely random playouts. To approximate the value of each $(s,a) \in \boldsymbol{X}$, the terminal reward from 1 random playout was used. A number of $\mathcal{R} = 60$ reward functions were sampled this way.

The experts were then defined as Softmax policies similar to the previous experiments, with low to zero noise for easier observation of results. The experts' value functions were derived from an approximative reward function sampled according to the above mentioned procedure (for a larger set of size $|\boldsymbol{X}| = 200$), but where some weights

Feature	Expert 1	Expert 2	Expert 3
Singlets `X`			1
Doublets `X`	1		
Triplets `X`			
Crosspoints `X`	1		
Corners `X`	1		1
Forks `X`		1	
Singlets `O`	1		
Doublets `O`	1		
Triplets `O`			
Crosspoints `O`	1		
Corners `O`	1		1
Forks `O`			1
Center occupation	1	1	1

Table 4.2: Factors of the reward weights $\boldsymbol{w}$ used for each expert's individual reward function $\rho_{\boldsymbol{w}}$. The base reward function was a random sample using the procedure described in the text. This method represents missing knowledge where the weights are 0 (empty table elements).

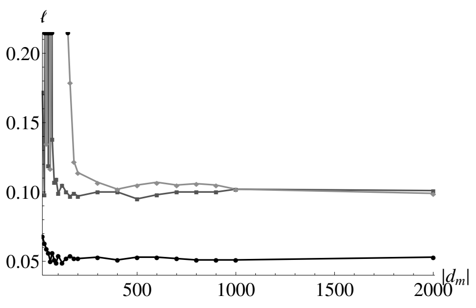
were set to 0 to simulate lack of information processing capabilities. For instance, if the weight for "Triplets `X`" were set to 0, the expert would not know how to win, but play optimally up until the last move (and then possibly win by pure chance). The purpose was to construct experts whose scoring were not too high, and the feature weights were chosen arbitrarily to achieve this. The factors of the weights per expert's reward function is presented in Table 4.2.

The expert's demonstrations $\boldsymbol{d}_m$ were used as samples for the LSTDQ policy evaluation step as explained in Section 3.4.

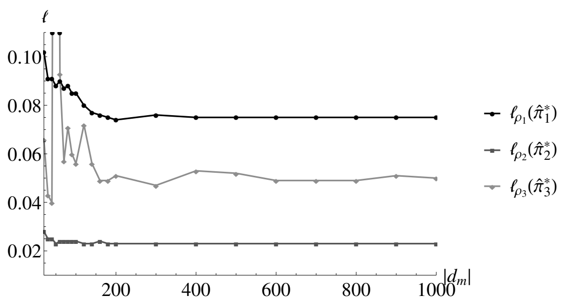
The convergence of the multitask experiment is presented in Figure 4.4. The performance results presented in Figure 4.5 is measured in terms of the *score* being the average reward $\in \{-1,0,1\}$ of random playouts from the initial (empty) state, facing an opponent that plays randomly unless it can win in 1 move. Here it is shown that it is possible for the derived policy $\widehat{\pi}^*$ to outperform those inferred from the individual experts.

The following other experiments, explained only briefly, involved different methods of sampling reward functions and different construction of expert policies that did not do as well.

Retaining only one dimension of the sampled reward function was tested, hoping that this would allow for further combinations within each task m and also between tasks.

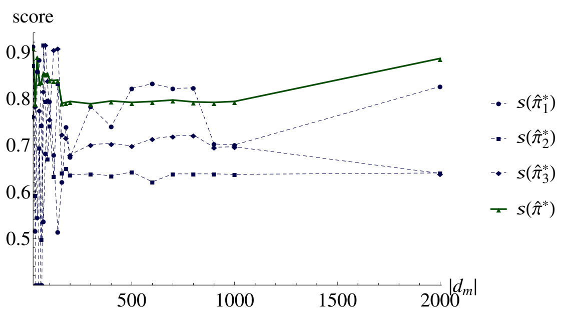


(a) First run

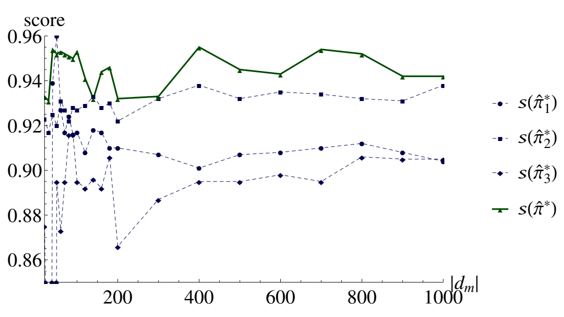


(b) Second run

Figure 4.4: Convergence of the loss of each inferred policy $\widehat{\pi}^*_m$ in the true m:th environment ρ_m.



(a) First run



(b) Second run

Figure 4.5: The solid green score $s(\widehat{\pi}^*)$ on top is the optimal policy derived from $\widehat{\rho} = \frac{1}{M}\sum_{m=1}^{M}\widehat{\rho}_m$, showing that the combined policy can outperform each individually inferred optimal policy $\widehat{\pi}^*_m$.

However, many of them resulted in nonsense optimal policies. For instance, if only the *singlets* feature was retained, the policy would avoid making doublets since that would eliminate the singlet. Also tested was to use the same Dirichlet sampling procedure as in Section 4.1, but since this method had litte correspondence to actually play the game, it could not be used. The prior in Section 3.5 was developed specifically for this purpose, and allows for weights $w_i \in (-\infty,\infty)$.

Optimal experts using values from the weights from Table 4.1 were tested, but only resulted in too fast convergence; the reward functions in $\mathcal{R}$ that best explained the real game would be inferred as most likely, and the corresponding optimal policies were optimal given only little amounts of data. Instead, learning from *imperfect* teachers was better for the evaluation of the proposed algorithm. Also, controlling the inference procedure was easier if the experts were defined in terms of a reward function rather than the value function.

5

Conclusions

The output of the algorithm is used to construct a deterministic policy which does not change beyond the early stages in the small MDP with 20 states and 5 actions. Although the posterior policy samples converge to the demonstrator's policies (not shown), the resulting deterministic policy is still identical when adding more data.

In the first experiments, each reward function is sampled from the simplex and all experts use the same reward function with different noise. This gave a loss $\ell_\rho(\widehat{\pi}^*)$ of the policy derived from $\widehat{\rho} = \frac{1}{M}\sum_{m=1}^{M}\widehat{\rho}_m$ which does not improve on the best loss in $\{\ell_\rho(\widehat{\pi}^*_m)\}_{m=1}^{M}$. The loss $\ell_\rho(\widehat{\pi}^*)$ is essentially an average across the losses of the indidivual experts. In other words, it seems as though, in practice, the performance is bounded by that of the best $\rho \in \mathcal{R}$.

A reason not to expect further variations when adding more data is that, in the limit of increased number of demonstrations or increased demonstration length, each sample from the policy posterior are identical points. This effectively lowers the importance of the number of Monte Carlo samples K, s.t. the BMTIRL approximation in (2.37) becomes a mean of of K identical values.

The model was quite sensitive to parameter tuning and since its inner workings are based on loss calculations, domain specific knowledge of the value function magnitude is required. The choice of norms ℓ_2 or ℓ_∞ also had a large effect on what values appeared in the loss matrix; the variation in magnitude of the supremum norm (the latter of the two) can be hard to anticipate. A choice to switch to ℓ_2 was made since it made observing the algorithm's behaviour easier, but if high dimensionality feature vectors are to be considered, this choice may need revision.

When moving to the true *multitask* setting, where there are M reward functions instead of only 1, it is possible to surpass the experts within environment ρ. This required the assumption that each ρ_m is generated with mean ρ. Similarly, the performance of the combined reward function for Tic-tac-toe outperforms that of each individual expert since the same assumptions hold here also.

5.1 Discussion and Future Work

Reinforcement learning techniques are very modular, and the contributions made here have mostly been to evaluate this modularity with large state spaces as a main consideration. The project involved the combination and implementation of work from several different papers, and much time was spent on evaluating different techniques.

The enumeration of reward functions and repeated solving of the reinforcement learning problem associated with calculating value functions for reward-policy pairs is slow and a potential problem in larger domains. BMTIRL would have to be modified on a greater extent than was done here to fully utilize knowledge of values near terminal states for improved sampling. It would benefit to develop methods for calculating the posterior reward distribution based on a parameterisation rather than performing discrete enumeration.

Inference was made possible in large domains thanks to the combination of the chosen policy space and the fact that LSTDQ to some extent mitigated having observations in only a small part of the state space. However, it is not clear how this scales to more complex domains, where trajectories are more spread apart and an insufficient set of features will have a much larger impact on the ability to differentiate between policies. Thus, future work would involve a focus on feature engineering and the development of a feature based policy prior that does not depend on state-action counts. It is tempting to use the Softmax policy prior but it does not allow for a closed form posterior calculation.[1]

An interesting observation during the construction of the experts was that when combining two mutually exclusive (in weights) reward functions, the optimal policy of the combined reward functions could in some cases have *worse* performance than those of its individual parts. This relates to the discussion in Section 2.4 about the sensitivity of the construction of reward functions. Future work in this area would involve further investigation on how the combination of reward functions affects the resulting optimal policies. This is necessary if one wants to take advantage of the value function relationship assumed in Figure 3.1.

The Tic-tac-toe results motivate the development of more complex generative models for how the experts choose their reward functions; there should be alot of structure and they are not independent, there might be clusters and so on. This is certainly one of the more interesting paths for future work.

[1]A Softmax policy prior would on the other hand allow for using a MAP approach such as in [34] at the expense of not being fully Bayesian. For convexity and to likely be solvable by gradient based techniques it also requires that the prior on reward parameters is flat. In this model, the reward prior from Section 3.5 is Gaussian and can be combined with existing techniques that approximate state-action values.

Bibliography

[1] C. Dimitrakakis, C. A. Rothkopf, Bayesian multitask inverse reinforcement learning, in: Recent Advances in Reinforcement Learning, Springer, 2012, pp. 273–284.

[2] V. L. Allis, Searching for solutions in games and artificial intelligence, Ph.D. thesis, Universiteit Maastricht (1994).

[3] S. Calinon, Robot programming by demonstration, in: Springer handbook of robotics, Springer, 2008, pp. 1371–1394.

[4] B. D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robotics and autonomous systems 57 (5) (2009) 469–483.

[5] S. J. Pan, Q. Yang, A survey on transfer learning, Knowledge and Data Engineering, IEEE Transactions on 22 (10) (2010) 1345–1359.

[6] L. Chen, P. Pu, Survey of preference elicitation methods, Tech. rep., Technical Report IC/200467, Swiss Federal Institute of Technology in Lausanne (EPFL) (2004).

[7] C. A. Rothkopf, C. Dimitrakakis, Preference elicitation and inverse reinforcement learning, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2011, pp. 34–48.

[8] S.-l. Huang, Designing utility-based recommender systems for e-commerce: Evaluation of preference-elicitation methods, Electronic Commerce Research and Applications 10 (4) (2011) 398–407.

[9] M. Minsky, Steps toward artificial intelligence, Proceedings of the IRE 49 (1) (1961) 8–30.

[10] W. Schultz, Predictive reward signal of dopamine neurons, Journal of neurophysiology 80 (1) (1998) 1–27.

[11] W. Schultz, P. Dayan, P. R. Montague, A neural substrate of prediction and reward, Science 275 (5306) (1997) 1593–1599.

[12] R. S. Sutton, Learning to predict by the methods of temporal differences, Machine learning 3 (1) (1988) 9–44.

[13] A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.

[14] M. Friedman, L. J. Savage, The expected-utility hypothesis and the measurability of utility, The Journal of Political Economy (1952) 463–474.

[15] J. Wal, J. Wessels, Markov decision processes, Statistica Neerlandica 39 (2) (1985) 219–233.

[16] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, Vol. 414, John Wiley & Sons, 2009.

[17] R. Bellman, The theory of dynamic programming, Tech. rep., DTIC Document (1954).

[18] I. H. Witten, An adaptive optimal controller for discrete-time markov environments, Information and control 34 (4) (1977) 286–295.

[19] G. A. Rummery, M. Niranjan, On-line Q-learning using connectionist systems, University of Cambridge, Department of Engineering, 1994.

[20] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (3-4) (1992) 279–292.

[21] M. G. Lagoudakis, R. Parr, Least-squares policy iteration, The Journal of Machine Learning Research 4 (2003) 1107–1149.

[22] C. D. Meyer, Matrix analysis and applied linear algebra, Vol. 2, Siam, 2000.

[23] J. A. Boyan, Least-squares temporal difference learning, in: ICML, Citeseer, 1999.

[24] S. J. Bradtke, A. G. Barto, Linear least-squares algorithms for temporal difference learning, Machine Learning 22 (1-3) (1996) 33–57.

[25] A. Y. Ng, S. J. Russell, et al., Algorithms for inverse reinforcement learning., in: Icml, 2000, pp. 663–670.

[26] P. Abbeel, A. Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the twenty-first international conference on Machine learning, ACM, 2004, p. 1.

[27] A. Wilson, A. Fern, S. Ray, P. Tadepalli, Multi-task reinforcement learning: a hierarchical bayesian approach, in: Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 1015–1022.

[28] A. Lazaric, M. Ghavamzadeh, et al., Bayesian multi-task reinforcement learning, in: ICML-27th International Conference on Machine Learning, 2010, pp. 599–606.

[29] T. Heskes, Solving a huge number of similar tasks: A combination of multi-task learning and a hierarchical bayesian approach., in: ICML, Vol. 15, Citeseer, 1998, pp. 233–241.

[30] K. Crowley, R. S. Siegler, Flexible strategy use in young children's tic-tac-toe, Cognitive Science 17 (4) (1993) 531–561.

[31] W. Konen, T. Bartz-Beielstein, Reinforcement learning for games: failures and successes, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, ACM, 2009, pp. 2641–2648.

[32] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, D. B. Rubin, Bayesian data analysis, CRC press, 2003.

[33] P. Hennig, D. Stern, T. Graepel, Coherent inference on optimal play in game trees.

[34] A. C. Tossou, C. Dimitrakakis, Probabilistic inverse reinforcement learning in unknown environments, arXiv preprint arXiv:1307.3785.