Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Apprenticeship Learning

Author:
Thomas Edlich

Supervisor:
Aldo Faisal

September 2018

Abstract

Apprenticeship learning is concerned about learning a task from an expert instead of from scratch in an isolated environment as it is the case for reinforcement learning. This is particularly useful in cases where it is hard to mathematically encode a task in a reward function but easy to either demonstrate the task or judge an execution of it. An example applications is driving where it is hard to define rewards which specify what driving behaviour is optimal but demonstrating it is quite simple.

The field of apprenticeship learning can be divided into three major subfields. First of all imitation learning, a technique in which an agent observes an expert performing a task and then tries to mimic the behaviour. Intention learning on the other side attempts to understand the reasons behind the expert's decisions and then finding a strategy based on these extracted objectives. The last category is interactive learning which, in contrast to imitation and intention learning, does not learn from a fixed dataset of demonstrations but instead allows communication between learner and expert in the form of queries or feedback.

This report aims to provide an overview of apprenticeship learning techniques, their respective benefits and drawbacks. We furthermore demonstrate how apprenticeship learning can be applied in some technical experiments.

Contents

List of Figures . . . . . **ii**

1 Introduction **1**

2 Preliminaries and Definitions **3**

3 Apprenticeship Learning **5**
3.1 Imitation Learning 6
3.1.1 Supervised Imitation Learning 7
3.1.2 Generative Adversarial Imitation Learning 8
3.2 Intention Learning 9
3.2.1 Inverse Reinforcement Learning 9
3.2.2 Dealing with Non-Optimal Demonstrations 12
3.2.3 Non-linear Inverse Reinforcement Learning 16
3.3 Interactive Learning 18
3.3.1 Active Inverse Reinforcement Learning 18
3.3.2 Apprenticeship Learning from Human Preferences 20

4 Experiments **23**
4.1 Gridworld 23
4.2 Cart Pole 24

5 Conclusion **27**

Bibliography **28**

List of Figures

2.1 Agent-Environment Interaction 3

3.1 Goal of Apprenticeship Learning 5
3.2 Goal of Inverse Reinforcement Learning. 9
3.3 Apprenticeship Learning via Inverse Reinforcement Learning 10
3.4 General structure of the apprenticeship learning from human preferences approach (Christiano et al., 2017). 20

4.1 16 by 16 gridworld consisting of 64 macrocells. Coloring represents the reward. 24
4.2 Experimental comparison of different apprenticeship learning approaches. Performance is displayed relatively to the expert policy (green) and a random policy (red). All results are averaged over 10 runs. 24
4.3 Recovered rewards using the MLIRL method. Left: Original rewards. Right: Rewards recovered by MLIRL using 64 trajectories. 25
4.4 Comparison of the apprenticeship from human preferences approach versus a Deep Q-Network trained using the true environment rewards. 25

Chapter 1

Introduction

Learning is a completely normal and natural task for human beings, we do it basically constantly and in many cases even subconsciously. Machine learning has been trying to compete with human learning capabilities for years and has developed some mathematical concepts which try to imitate human perception and learning. Reinforcement learning is one such a concept. Herein an agent interacts with an environment and learns a strategy by trial-and error.

This kind of learning strategy is quite similar to human learning. In many cases, humans practice a task until they achieve a specific goal. One key difference however is that humans do not have a clear mathematical formulation of the task they want to learn or improve on, but only a vague idea of what they want to achieve. Furthermore, humans rarely learn to perform tasks, especially complicated task, just by trying it multiple times on their own. Instead, we often learn by watching other people like parents, teachers, coaches or professors, who can perform a task better than we or at least know what a good performance is.

The idea of learning by watching someone else execute a task has found its way into the machine learning field decades ago (see for example (Segre & DeJong, 1985), (Bakker & Kuniyoshi, 1996) and (Schaal, 1997)). The discipline of learning by observing a demonstration of a task is called apprenticeship learning or sometimes also learning from demonstration.

This report aims to give an overview of several apprenticeship learning methods, their use cases, benefits and drawbacks. After providing some preliminary definitions in section 2, we will start this overview with imitation learning (section 3.1), which was predominantly used in early years of learning from demonstration (see (Bain & Sammut, 1995) and (Schaal, 1999)). This approach essentially tries to memorize which actions the demonstrator performed in which situations and then blindly imitates this behaviour in order to solve a task on its own.

In section 3.2 we then present intention learning. In contrast to imitation learning, this approach does not attempt to blindly imitate an expert but instead tries to understand what the expert is doing and why he is doing it by extracting his object-

ives from the demonstrations and then using these objectives to find an own strategy.

Afterwards, we will conclude our overview of apprenticeship learning methods by presenting how an apprentice can learn in an interactive setting (section 3.3), that means when he is allowed to actively communicate with the demonstrator instead of passively observing. This approach comes even closer to human learning which usually heavily relies on asking questions and feedback.

In section 4 we will continue by showing how some of the presented apprenticeship learning methods can be applied and finally we will give a conclusion about apprenticeship learning and possible future directions of the field (section 5).

Chapter 2

Preliminaries and Definitions

In reinforcement learning, the goal is to teach an agent, either real or virtual, how to perform a certain task. The agent essentially interacts with the environment by executing actions and receiving a new state and a reward as a feedback from the environment (see Figure 2.1). An important mathematical concept in reinforcement learning are Markov Decision Processes (MDP). A MDP $\mathcal{M}$ is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, \gamma, D, R)$ where $\mathcal{S}$ denotes the state space; $\mathcal{A}$ the action space; $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition probabilities for a given state-action-state pair such that $T(s_1, a, s_2)$ is the probability of transitioning from s_1 to s_2 if action a is executed; $\gamma \in [0, 1]$ is called the discount factor; D is a start-state distribution over $\mathcal{S}$ and $R : \mathcal{S} \to \mathbb{R}$ is the reward function which specifies the reward an agent receives for reaching a given state.

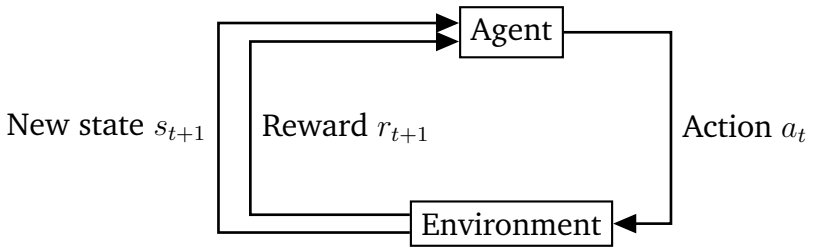

Figure 2.1: Agent-Environment Interaction

Apprenticeship learning is closely related to reinforcement learning, and therefore uses MDPs quite frequently. We will now state some key properties of MDPs. A policy is a mapping $\pi : \mathcal{S} \to \mathcal{A}$, which determines which action to execute in a certain state. The quality of a policy can be measured by its corresponding value function $V^\pi(s_0) : \mathcal{S} \to \mathbb{R}$ which is defined as the expected accumulated rewards when executing π starting at state s_0:

$$V^\pi(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] = R(s_0) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V^\pi(s') \tag{2.1}$$

The value of a state essentially measures how good it would be to encounter it. In order to be able to decide what actions to take in a given state, we additionally need a quality measure over states and action. This measure is called Q-function with $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. It is formally defined as:

$$Q^{\pi}(s,a) = R(s) + \gamma \mathbb{E}_{s' \sim T(s,a,s')}[V^{\pi}(s')] = R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s') \tag{2.2}$$

Reinforcement learning aims to find a policy which optimizes the expected reward. The Q-function tells us, how beneficial it is to take a certain action in a state. For given Q-values, the optimal policy π^* can then simply be found by:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s,a) \tag{2.3}$$

Many apprenticeship learning approaches require access to a reinforcement learning algorithm which is able to compute value and Q-function and the corresponding optimal policy. In the upcoming chapters, we will assume the availability of such a reinforcement learner. For more information about reinforcement learning algorithms see (Sutton & Barto, 1998).

Chapter 3

Apprenticeship Learning

Apprenticeship learning, often also called learning from demonstration, is closely related to reinforcement learning. The key difference is, that in apprenticeship learning the reward function is unknown. However, we are given recorded traces of an expert who is demonstrating the task which should be learned. The goal is then to find a policy which achieves equal performance as the expert with respect to the unknown reward function (Syed & Schapire, 2008). Figure 3.1 visualises this setting.

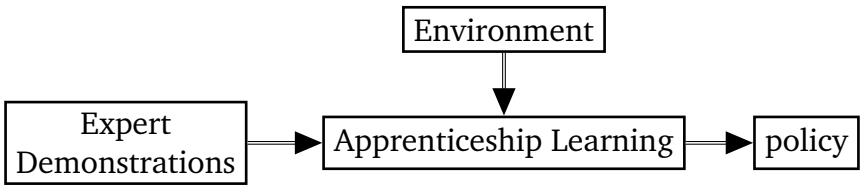

Figure 3.1: Goal of Apprenticeship Learning

One of the key components of reinforcement learning is the reward function since it compactly contains information about what states are desirable or not. In some settings, incorporating a goal into a mathematical notation arises rather naturally, for example when teaching an agent how to play chess a natural reward function would give a positive reward for winning, a neutral reward for a draw and a negative reward for losing. However, in many tasks it is quite difficult to find a clear mathematical definition for the reward function. One possibility to find a suited reward function is then to manually tweak the reward function until the expected behaviour is reached (Pieter Abbeel & Ng, 2004). Unfortunately, this task can be quite tedious. Consider, for example, the task of autonomous driving. It is basically impossible to manually define a reward function which comprehends a good representation of the task ”driving well”. This is mainly because, driving well contains several different components like not colliding with any obstacle, obeying traffic rules, reaching the destination fast enough, driving eco-friendly and driving in such a manner, that all passengers feel safe and comfortable, and many more. All these components would need to be manually incorporated and weighted in order to find a reward function which enables a reinforcement learning agent to learn to drive autonomously. In

contrast, it is fairly easy for a human to demonstrate the desired task which makes apprenticeship learning a useful option for such scenarios where reward functions are difficult to define manually.

Another possible scenario in which apprenticeship learning is helpful is as pre-training for a reinforcement learner. During training time, reinforcement learning agents usually have an extremely bad performance, since the agent needs to explore the state space in order to find out what a useful strategy would be. In some fields, this bad performance might also have a physical impact on the agent especially in robotics, where the robot might get damaged or damage something else by executing random actions. In these cases, one can apprenticeship learning to give the agent a head start in training. The idea behind this is, to first teach the agent a policy which mimics the harmless and goal-oriented actions of the demonstrator and then use Reinforcement Learning to further enhance the policy. (Hester et al., 2018) demonstrated such an approach which leverages expert demonstration as pre-training for a Deep Q-Network. Using this approach they managed to outperform state-of-the-art reinforcement learning algorithms in 11 of 42 Atari games. Furthermore, they showed that their algorithm had much better performance during the beginning of training than other Reinforcement Learning algorithms (Hester et al., 2018). This better start performance is key to applying Reinforcement Learning in areas where the learning cannot be done in a simulator but has to be performed within the real environment.

In the following sections, we will describe several approaches of implementing learning from demonstration. In general we can categorize apprenticeship learning in three main areas: imitation learning, intention learning and interactive learning. In imitation learning the agent attempts to mimic the expert's demonstrations while in intention learning we aim to first extract objectives from the demonstrations in order to infer what the general goal is and then search for a policy which achieves these goals. Interactive learning on the other hand allows communication between agent and expert, for example in the form of queries or feedback. First we will explain how to implement imitation learning and afterwards we will present several intention learning and interactive learning approaches.

3.1 Imitation Learning

Imitation learning, or often also called behavioural cloning, directly implements apprenticeship learning by mimicking the expert's demonstrations. For all states which have been encountered during demonstrations, this can easily be done. However, demonstrations usually only cover a fraction of the state space, such that we require some generalisation mechanism in order to choose useful actions in states which have not been visited during demonstration.

3.1.1 Supervised Imitation Learning

One fairly straightforward method of implementing imitation learning is using supervised learning. This approach tries to mimic the expert's behaviour by learning a mapping from a state to a specific action (Bain & Sammut, 1995). To extract this mapping from given expert trajectories $\mathcal{D} = \{\tau_1, \ldots, \tau_n\}, \tau_i = \{(s_0, a_0), \ldots, (s_t, a_t)\}$ we can use any supervised learning algorithm such as decision trees, support vector machines or neural networks. Two applications of this technique will be described below.

(Sammut, Hurst, Kedzier & Michie, 1992) showed how behavioural cloning can be used to generate an autopilot for a simulated aircraft. To accomplish this, they first collected several state-action trajectories by recording humans flying an aircraft in a simulator for a predetermined route. To extract rules for the autopilot, the authors used decision trees. While their generated autopilot was indeed able to produce similar flight trajectories as the human demonstrators, the authors noted that the system was solely based on reactive decisions, since no representation of a goal was derived. This resulted in fairly unstable and non-smooth flight manoeuvrers (Sammut et al., 1992).

Recent success of reinforcement learning on Atari games (see (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015) and (Mnih, Kavukcuoglu, Silver, Graves et al., 2013)) has inspired (Bogdanovic, Markovikj, Denil & De Freitas, 2015) to examine the applicability of apprenticeship learning for this kind of tasks. Their method "Deep Apprenticeship Learning" (DAL) aims to imitate the expert's behaviour while playing an Atari game by learning the state-action mapping using a deep convolutional neural network. This network is fairly similar to Deep Q-Networks which were used by (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015; Mnih, Kavukcuoglu, Silver, Graves et al., 2013) in order to teach reinforcement learning agents to play Atari games. As input to the network, the authors used stacked pixel matrices of the current and some previous game states. The DAL network is then supposed to predict an action for the given state input. While DQNs are trained to maximize a reward function, the DAL network is trained in a supervised fashion, where the input is a game state the human expert encountered during play and the target output is the corresponding action the demonstrator executed. After training, this network is then able to autonomously play Atari. The experiments performed by (Bogdanovic et al., 2015) show that their algorithm is able to produce a reasonably good strategy which is however inferior to the human players the network learned from.

Behavioural cloning via supervised learning is in general a fairly easy solution to apprenticeship learning since it reduces the task to a classification task. However, this technique usually requires the demonstrations to have a reasonable coverage of the state space. Otherwise the trained agent will perform badly in state regions where few or none exploration has taken place.

3.1.2 Generative Adversarial Imitation Learning

Another, more sophisticated way of implementing imitation learning is to use adopt a generative-adversarial setting. This approach is based on the idea of generative adversarial neural networks (GANs). GANs have proven to be very effective in approximating a data distribution $p(X)$ (see for example: (Huang, S. Zhang, Li, He et al., 2017; H. Zhang et al., 2017)). They achieve this by using a discriminator and a generator neural network which compete against each other. More formally, the discriminator network D is trained to maximize the probability of correctly labelling data points which are either drawn from the training data or generated by G. Meanwhile, G is trained to minimize $\log(1 - D(G(\boldsymbol{z})))$ and therefore essentially aims to generate realistic-looking data points which fool D. This whole training process can more compactly be compressed as a two-person minimax game with the following value function (Goodfellow et al., 2014):

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log 1 - D(G(\boldsymbol{z}))], \tag{3.1}$$

where $p_{data}(\boldsymbol{x})$ denotes the original data distribution and, $\boldsymbol{z}$ the random input for G and $p_{\boldsymbol{z}}(\boldsymbol{z})$ the prior on $\boldsymbol{z}$.

In order to use such a setting for imitation learning, we need a discriminator, whose goal it is to distinguish if for a given state s an action a has been artificially generated or if it has been drawn from the expert's policy and we need a generator which on the other side attempts to generate an action a for a given state s such that it fools the discriminator. One approach which adopts this idea is Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016). This algorithm works similarly to a generative adversarial network. However, instead of using a neural network to model the generator G they use a parametrized policy $\pi_t heta$. Equation (3.1) can then be rewritten as:

$$\min_\theta \max_D \mathbb{E}_{\pi_E}[\log D(s,a)] + \mathbb{E}_{\pi_\theta}[\log 1 - D(s,a)] - \lambda H(\pi_\theta), \tag{3.2}$$

where $H(\pi) = \mathbb{E}[-\log \pi(a|s)]$ is defined as the γ-discounted causal entropy of π (see (Bloem & Bambos, 2014)) and serves as a regularizer (Ho & Ermon, 2016). Equation (3.2) can then be solved iteratively. In each iteration we start by sampling some trajectories from the current policy and train the discriminator D using these sampled trajectories and the expert's traces and therefore maximise Equation (3.2) with respect to D. Afterwards, we take a policy optimisation step and therefore update θ_i to θ_{i+1} in order to minimise Equation (3.2) with respect to π. The authors used Trust Region Policy Optimization (TRPO (Schulman, Levine, Abbeel, Jordan & Moritz, 2015)) for the policy update, since this method prevents the policy of changing too much in one step due to noise in the policy gradient (Ho & Ermon, 2016). The whole GAIL algorithm is furthermore depicted in Algorithm 1.

(Ho & Ermon, 2016) tested their algorithm on several reinforcement learning tasks, including classical tasks like mountain car (Moore, 1990) or cartpole (Barto, Sut-

Algorithm 1 Generative Adversarial Imitation Learning (GAIL)

Require: Expert trajectories $\zeta_{E,i} \sim \pi_E$, initial policy parameters θ_0, discriminator parameters $\boldsymbol{w}_0$
  for $i = 0, 1, 2, \ldots$ **do**
    Sample trajectories $\zeta_{G,i} \sim \pi_{\theta_i}$
    Update discriminator parameters from w_i to w_{i+1}
    Update policy parameters from θ_i to θ_{i+t}
  end for

ton & Anderson, 1983), and on the other hand high-dimensional control task for 3D humanoid locomotion. In most tasks GAIL was able to significantly outperform supervised behavioural cloning and furthermore often achieved performances comparable to the expert (Ho & Ermon, 2016). An exemplary application of GAIL to driving scenarios has been demonstrated in (Kuefler, Morton, Wheeler & Kochenderfer, 2017).

3.2 Intention Learning

In contrary to imitation learning (see section 3.1) which extracts a policy directly from the given demonstrations, intention learning aims to extract the expert's objectives, essentially trying to infer what the general goal of the task is, and then uses the extracted intentions to search for a policy which achieves these. This has the interesting consequence that the final policy found by intention learning, in contrast to imitation learning, might differ even in seen states from the expert's policy.

3.2.1 Inverse Reinforcement Learning

In reinforcement learning, we search for a policy which maximizes a reward function. These rewards essentially embed the goal of a task. The task of inferring the rewards from a policy instead of the other way around, is therefore called inverse reinforcement learning (S. Russell, 1998). In most cases, we do not have access to the complete policy, but only to traces of expert's demonstrations. Figure 3.2 shows the general inverse reinforcement learning setting.

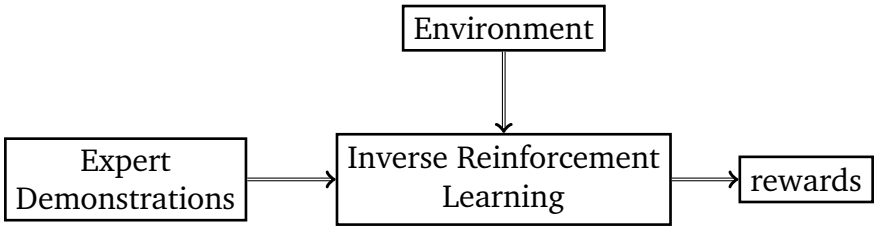

Figure 3.2: Goal of Inverse Reinforcement Learning.

Inverse reinforcement learning can essentially be used for two different purposes. First of all, it can be applied in order to extract the objectives an individual has during performing a task. The idea behind this is, that humans always subconsciously try to maximize an internal reward function whenever they perform a task. Finding a mathematical representation of this inner reward function can then reveal what motivated the human to execute a certain action in a specific situation. This report however focusses on the second possible use case of inverse reinforcement learning, that is apprenticeship learning. This essentially works by first extracting a reward function from the expert's demonstrations and then leveraging traditional reinforcement learning techniques to find an optimal policy for this reward function (see Figure 3.3).

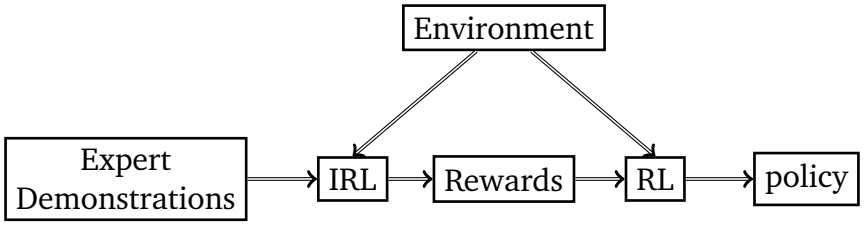

Figure 3.3: Apprenticeship Learning via Inverse Reinforcement Learning

In this section we will present several approaches of how apprenticeship learning can be implemented using inverse reinforcement learning. Beforehand, we would like to introduce some notations. Inverse reinforcement learning takes as input a Markov Decision Process (MDP) without a reward function, we denote this by $\mathcal{M} \backslash R$. Furthermore, we are given a set $\mathcal{D} = \{\tau_1, \ldots, \tau_n\}$ of $n \geq 1$ expert demonstration trajectories, where each trajectory is a sequence of state-action pairs, such that $\tau_i = ((s_0, a_0), \ldots, (s_t, a_t))$.

The key issue in inverse reinforcement learning is reward degeneracy, i.e. that a large set of reward functions exists for given expert traces or even for a given complete policy (Ng, S. J. Russell et al., 2000). First of all, any multiple of the true rewards give the same solution. This is however not a real issue, since the correct reward structure could still be recovered. Furthermore, constant reward function ($R(s) = c, c \in \mathbb{R}$) are always a solution for the inverse reinforcement learning problem (proven in (Ng, S. J. Russell et al., 2000)). Since in this case, the agent achieves the same reward no matter what action is taken, every policy is optimal, including the policy presented by the expert. Therefore, inverse reinforcement learning algorithms choose different criteria for which they compute the optimal reward function. In the next sections, we will present several approaches to inverse reinforcement learning algorithms and how they decide on which reward function to choose.

One of the first proposed solutions for implementing apprenticeship learning by using inverse reinforcement learning was proposed by (Pieter Abbeel & Ng, 2004).

Their first key assumptions is that the expert behaves optimally for the given task according to an unknown policy π_E. Furthermore, they assume that the reward for a state is a linear function of certain known state features $\phi(s) : \mathcal{S} \rightarrow [0,1]^k$, such that we can formalize the reward function for a state $s \in \mathcal{S}$ as follows:

$$R(s) = \boldsymbol{w}^\mathsf{T}\phi(s), \tag{3.3}$$

where $\boldsymbol{w} \in \mathbb{R}^k, \|w\|_1 \leq 1$. The true unknown reward function is denoted as $R^*(s) = \boldsymbol{w}^{*\mathsf{T}}\phi(s)$. They furthermore defined the notion of *feature expectations* μ of a policy π:

$$\mu(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]. \tag{3.4}$$

This essentially describes the expected accumulated discounted features which occur when executing actions according to π. The expert's feature expectations can be estimated using the n sample trajectories by:

$$\mu_E = \hat{\mu}(\pi_E) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{n} \sum_{t=0}^{|\tau_i|-1} \gamma^t \phi(s_t^i), \tag{3.5}$$

Using this feature expectation notation we can reformulate the value of a state s_0 given a policy π as follows:

$$V^\pi(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] = \boldsymbol{w}^\mathsf{T}\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] = \boldsymbol{w}^\mathsf{T}\mu(\pi). \tag{3.6}$$

The fundamental idea of (Pieter Abbeel & Ng, 2004) is that if we can find a policy which has approximately equal feature expectations as the expert's policy then the expected reward will be approximately equal as well. More formally, we are attempting to find a policy $\hat{\pi}$ with $\|\mu(\pi^*) - \mu(\hat{\pi})\|_2 < \epsilon$ since then the expected rewards of π and $\hat{\pi}$ will also differ by at most ϵ (Pieter Abbeel & Ng, 2004).

The proposed algorithm by Abbeel et al. is shown in Algorithm 2. The key step in this algorithm is the optimisation problem in Step 4. Herein, we compute reward weights w_i for which the expert would maximally outperform the best policy for these reward weights. This step is essentially a inverse reinforcement learning step which finds the reward weights for which the expert outperforms all previously found policies by the largest margin. The calculations are terminated once we cannot find a reward function which outperforms any policy by at least ϵ, since this guarantees that there is at least one policy $\pi_i \in \Pi$ which achieves approximately the same expected rewards as π_E for the unknown reward function R^* (Pieter Abbeel & Ng, 2004). To choose a good policy from the returned set, (Pieter Abbeel & Ng, 2004) propose to either ask a human expert to inspect the policies and then pick the most appropriate one or to generate a mixed policy from the set by building a convex combination from all found policies in order to minimize the difference of feature expectations (Pieter

Abbeel & Ng, 2004).
The feature matching algorithm has been successfully applied to several real-world problems, including autonomous helicopter control (Coates, Abbeel & Ng, 2009) and motion planning (P. Abbeel, Dolgov, Ng & Thrun, 2008).

Algorithm 2 Feature Matching

generate random policy π_0 and compute $\mu(\pi_0)$.
$i = 1$
while True **do**
    Compute t_i as: $t_i = \max_{w:\|w\|_2} \min_j \boldsymbol{w}^\mathsf{T}(\mu_E - \mu(\pi_j))$
    Let $\boldsymbol{w}_i$ be the value that attains t_i
    if $t_i < \epsilon$ **then**
        break
    end if
    Using an RL algorithm find optimal policy π_i for $R(s) = \boldsymbol{w}_i^\mathsf{T}\phi(s)$.
    Compute $\mu(\pi_i)$
    $i = i + 1$
end while
return $\Pi = \{\pi_0, \pi_1, \ldots, \pi_i\}$

3.2.2 Dealing with Non-Optimal Demonstrations

In the previous section we described how Feature Matching finds a policy which has the same feature expectations as the expert demonstrations. Therefore the performance is approximately equal to the expert as well. This however means, that the apprentice's performance is upper- and lower-bounded by the expert (Syed & Schapire, 2008) such that suboptimal demonstrations lead to a suboptimal agent. In general, human performances are not optimal since humans might not know the perfect behaviour and human actions always contain noise. Assuming expert optimality is therefore unrealistic. In the following we will present several inverse reinforcement learning approaches which drop this assumption in order to deal with suboptimal task demonstrations.
One such approach was proposed by (Syed & Schapire, 2008). The authors pose the inverse reinforcement learning problem as a two-player zero-sum game in which the apprentice competes against the environment. This competition works by letting the apprentice choose a policy to maximise its performance relative to the expert while the environment chooses a reward function adversarially in order to minimise the apprentice's performance. Similarly to Feature Matching (see 3.2.1) this approach assumes that the reward is a linear function of state features, such that $R(s) = \boldsymbol{w}^\mathsf{T}\phi(s)$, however they assume that $\phi(s) \in [-1, -1]^k$ and $\boldsymbol{w} \in \{\boldsymbol{w} \in \mathbb{R}^k : |\boldsymbol{w}|_1 = 1, \boldsymbol{w} \succeq 0\}$. The game can then be formally defined as follows:

$$v^* = \max_{\pi} \min_{\boldsymbol{w}} \boldsymbol{w}^\mathsf{T}\mu(\pi) - \boldsymbol{w}^\mathsf{T}\mu_E \tag{3.7}$$

To find the game value v^* (Syed & Schapire, 2008) use the multiplicative weights algorithm (Freund & Schapire, 1999) since it can handle huge or even unknown game matrices in contrast to the traditional linear programming approach for solving such games.

(Syed & Schapire, 2008) proved that their Multiplicative Weights for Apprenticeship Learning (MWAL) returns policy which is also lower-bounded by the expert's performance. However, they also proved, that their algorithm might outperform the expert. The reason for this lies in their assumptions about the features $\phi(s)$. In contrast to Feature Matching, these features can also be negative, while all weights are assumed to be non-negative. Therefore, the apprentice knows in advance whether a feature positively or negatively influences the reward although the actual reward function is unknown. MWAL therefore enables us to incorporate more prior knowledge into the features (Syed & Schapire, 2008).
Expert's suboptimality can also be dealt with in a probabilistic way. This is usually achieved by modelling the expert's action choice by a Boltzmann distribution which assigns an action a probability exponentially proportional to its Q-value. This essentially assumes that an expert is way more likely to pick good actions than really bad actions, but might not always choose the optimal action. In order to incorporate our prior beliefs in the expert's competence, we also add a multiplicative factor α:

$$P((s,a)|R) = \frac{1}{Z}\exp\left(\alpha Q(s,a,R)\right), \tag{3.8}$$

where Z is a normalizing constant. Using this distribution we can derive several methods to find a suitable reward function. One such approach is Maximum Likelihood Inverse Reinforcement Learning (MLIRL) (Babes, Marivate, Subramanian & Littman, 2011; Vroman, 2014). MLIRL again assumes that the rewards linearly depend on state features, such that $R(s) = \boldsymbol{w}^\mathsf{T}\phi(s)$. In order to derive the maximum likelihood solution we need to maximize the following loss function with respect to the weights $\boldsymbol{w}$:

$$L(\mathcal{D}|\boldsymbol{w}) = \log P(\mathcal{D}|\boldsymbol{w}) = \sum_{\tau\in\mathcal{D}} w_\tau \sum_{(s,a)\in\tau} \log P((s,a)|\boldsymbol{w}), \tag{3.9}$$

where w_τ is an optional weight associated with trajectory τ. Maximizing Equation (3.9) requires that L is differentiable with respect to $\boldsymbol{w}$. To achieve this (Vroman, 2014) modify the definition of Q-values (see Equation (2.2)) to:

$$\begin{aligned} Q(s,a,\boldsymbol{w}) &= R(s) + \gamma \sum_{s'\in\mathcal{S}} T(s,a,s') \otimes'_a Q(s,a',\boldsymbol{w}) \\ \otimes'_a Q(s,a',\boldsymbol{w}) &= \frac{\sum_{a'} Q(s,a')\exp(\alpha Q(s,a'))}{\sum_{a''}\exp(\alpha Q(s,a''))}. \end{aligned} \tag{3.10}$$

To find the optimal solution $\boldsymbol{w}^*$ we can now use gradient ascent or any other optimisation method.
Another probabilistic approach for inverse reinforcement learning is Bayesian In-

verse Reinforcement Learning (BIRL) (Ramachandran & Amir, 2007). While MLIRL computed a maximum likelihood solution, BIRL derives a posterior distribution over rewards in order to handle suboptimal demonstrations. This posterior can be derived by applying the Bayes rule:

$$P(\boldsymbol{R}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{R})P(\boldsymbol{R})}{P(\mathcal{D})} \tag{3.11}$$

In this case $\boldsymbol{R} \in \mathbb{R}^{|\mathcal{S}|}$ denotes a reward vector with $\boldsymbol{R}_i = R(s_i)$. BIRL is therefore unfortunately restricted to cases where all states are known and can be enumerated. However, we drop the assumption of a linear reward function as in MLIRL and Feature Matching. The distribution of (s, a) given $\boldsymbol{R}$ is again assumed to be a Boltzmann distribution (see Equation (3.8)). However, in this case the Q-value computations do not need to be modified, such that we compute $Q(s, a, \mathbb{R})$ as in Equation (2.2).

Assuming that the expert's policy is stationary (i.e. invariant with respect to time), which implies that each action choice is independent of all previous decision, then we write the likelihood of our evidence $\mathcal{D}$ as:

$$P(\mathcal{D}|\boldsymbol{R}) = \prod_{\tau\in\mathcal{D}} \prod_{(s,a)\in\tau} P((s,a)|\boldsymbol{R}) = \frac{1}{Z}\exp(\alpha \sum_{\tau\in\mathcal{D}} \sum_{(s,a)\in\tau} Q^*(s_{ij}, a_{ij}, \boldsymbol{R})). \tag{3.12}$$

The reward posterior can then be derived by:

$$P(\boldsymbol{R}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{R})P(\boldsymbol{R})}{P(\mathcal{D})} = \frac{1}{Z'}\exp\left(\alpha \sum_{i=1}^{m}\sum_{j=0}^{T_m} Q^*(s_{ij}, a_{ij}, \boldsymbol{R})\right) P(\boldsymbol{R}). \tag{3.13}$$

The choice of a prior $P(\boldsymbol{R})$ depends on the kind of task we are attempting to learn. In cases where no preference over rewards is available, a uniform prior between R_{min} and R_{max} can be used. Most real world problems however have many states with no or negligibly small rewards and few states with unusually high or low rewards. For these cases, a Gaussian or Laplacian prior is more suitable. Furthermore, for planning problems with positive rewards for goal states and negative rewards for other states, can be better modelled by a Beta distribution (Ramachandran & Amir, 2007).

In apprenticeship learning, the goal is to find a policy which yields similar performance to the demonstrations by an expert. This can formally be defined by a policy loss:

$$L^p(\boldsymbol{R}, \pi) = \|\boldsymbol{V}^*(\boldsymbol{R}) - \boldsymbol{V}^\pi(\boldsymbol{R})\|_p, \tag{3.14}$$

where p denotes an arbitrary norm. Since BIRL is a probabilistic approach, we wish to find π which minimizes the expected policy loss $\mathbb{E}_{P(R|\mathcal{D})}[L^p(\boldsymbol{R}, \pi)]$. Computing this minimization is however hard due to the normalization constant in Equation (3.13). To solve this issue, (Ramachandran & Amir, 2007) showed that the policy π which minimizes the expected policy loss is equivalent to the optimal policy for the given MDP$\backslash\boldsymbol{R}$ extended by the expected reward vector $\mathbb{E}[\boldsymbol{R}]$. We therefore now need to find $\mathbb{E}[\boldsymbol{R}]$ and then can use these rewards to find π via reinforcement learning. In

Algorithm 3 Policy Walk

choose random reward vector $\boldsymbol{R} \in \mathbb{R}^{|S|}/\delta$
$\pi = \texttt{PolicyIteration}(\mathcal{M}, \boldsymbol{R})$
repeat
    Choose $\boldsymbol{R}'$ from neighbours of $\boldsymbol{R}$
    Compute $Q^{\pi}(s, a, \boldsymbol{R}')$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
    if $\exists (s, a) \in \mathcal{S} \times \mathcal{A} : Q^{\pi}(s, \pi(s), \boldsymbol{R}') < Q^{\pi}(s, a, \boldsymbol{R}')$ **then**
        $\pi' = \texttt{PolicyIteration}(\mathcal{M}, \boldsymbol{R}', \pi)$
        With probability $p = \min 1, \frac{P(\boldsymbol{R}', \pi')}{P(\boldsymbol{R}, \pi)}$: Set $R = R'$ and $\pi = \pi'$
    else
        With probability $p = \min 1, \frac{P(\boldsymbol{R}', \pi)}{P(\boldsymbol{R}, \pi)}$: Set $R = R'$
    end if
return $\boldsymbol{R}$

order to compute this, (Ramachandran & Amir, 2007) proposed a MCMC sampling algorithm called `PolicyWalk` (see Algorithm 3).

There are several more probabilistic approaches for inverse reinforcement learning besides MLIRL and BIRL. Examples of these include Maximum Entropy inverse reinforcement learning (Ziebart, Maas, Bagnell & Dey, 2008) which defines a probability distribution over trajectories where the probability of a single trajectory is proportional to the exponentially expected reward for this trajectory. And furthermore, (Choi & Kim, 2011) extend BIRL to a maximum a posteriori (MAP) estimator for rewards using a gradient method.

So far we have assumed that the demonstrations are non-optimal due to the expert making mistakes or choosing suboptimal actions. However, expert behaviour varies not only in their performance but possibly also in their intentions, meaning that the expert in fact performs the same task but pursues different goals and therefore deliberately makes different choices. Considering the task of driving, at one time, the demonstrator might be late and therefore values speed higher than usual, at another time, he might drive extra carefully because he is transporting fragile goods. Obviously, both behaviours cannot be captured by the same reward function and learning a single reward function from such noisy demonstrations could possibly result in bad apprentice performance. (Babes et al., 2011) present a method to deal with such a setting wherein the expert's demonstrations are generated with multiple intentions. The main idea of their approach is to cluster the given trajectories using a Expectation-Maximization method.

The proposed method requires the number K of intentions in the set of trajectories $\mathcal{D} = \{\tau_1, \ldots, \tau_N\}$ to be known. The clustering algorithm starts by initialising rewards w_i and prior probabilities ρ_i for each cluster. In order to cluster the trajectories, the E-Step computes the probability z_{ij} that trajectory i belongs to cluster j:

$$z_{ij} = \prod_{(s,a)\in\tau_i} P((s,a)|\boldsymbol{w})\frac{\rho_j}{Z}, \tag{3.15}$$

where $P((s,a)|\boldsymbol{w})$ is given as in Equation (3.8) and Z is a normalization constant. In the M-Step, the parameters w_j and ρ_j are updated for each cluster i. This is done by setting $\rho_j = \sum_i \frac{z_i j}{N}$ and computing reward weights w_j using MLIRL with weight z_{ij} for trajectory τ_i. Instead of MLIRL, any other inverse reinforcement learning method could however be used.

3.2.3 Non-linear Inverse Reinforcement Learning

In most previous methods we have assumed that the reward is a linear function state features. This is however a pretty strong restriction. In order to apply inverse reinforcement learning methods which are based on this assumption to complicated problems, we usually need to manually construct meaningful state features such that the reward can be modelled as a linear function of these features (similar to linear regression). This can be however very tedious such that methods which learn non-linear reward functions are desirable. In the following, we will give an overview of such methods.

Constructing meaningful features for linear inverse reinforcement learning can be hard. However, it is often easier to generate many base features, i.e. simple, non-explanatory features. These features may individually not be very meaningful but by combining several together, more significant features could be found. The Feature Construction for Inverse Reinforcement Learning (FIRL) approach by (Levine, Popovic & Koltun, 2010) builds upon this idea. This method essentially takes a collection of component features as an input and then automatically constructs more sophisticated features as logical conjunctions of these. The final reward function is then assumed to be linear in the constructed features, but it is non-linear in the original features.

Starting with an empty feature set Φ_0 the feature construction is done iteratively. Each iteration consists of an optimisation step, in which the algorithm searches for a reward function R_i for the current feature set Φ_{i-1}, and a fitting step which updates the feature set. The reward function in the optimisation step is hereby chosen such that it is compliant with the demonstrations by the expert and furthermore minimizes the sum of squared errors between R_i and its projection to the the linear basis of the current feature set Φ_{i-1}. In the fitting step the feature set is updated by building a regression tree on the state space $\mathcal{S}$ for R_i, where the component features are the nodes. Each state is therefore uniquely associated with one leaf node. Assume that this regression tree has ℓ leaf nodes (denoted by ϕ_i), meaning that we have ℓ different reward values. The new features are then indicator functions $f_i(s) = \mathbb{I}[s \in \phi_i]$ (Syed & Schapire, 2008). After each iteration, the current reward R_i can therefore be expressed as a linear combinations $\Phi_i = \{f_1, \ldots, f_\ell\}$.

While the FIRL method solves the problem of manual feature construction, it unfortunately assumes expert optimality due to the fact that it chooses reward function in the optimisation step which is consistent with the demonstrations. As discussed earlier, optimal demonstrations are usually not available. Therefore, other inverse reinforcement learning methods which are able to handle both non-linear reward functions and suboptimal demonstrations are needed.

One such approach is Gaussian Process Inverse Reinforcement Learning (GPIRL) which was introduced by (Levine, Popovic & Koltun, 2011). Assume for now, that besides the expert demonstrations we are additionally given the true rewards for the encountered states but the rewards for all other states are unknown. The problem of inferring the rewards for new states from given sample points essentially induces a regression problem for which Gaussian processes (GPs) can be used (for more information on GP regression refer to (Rasmussen, 2004)). Unfortunately, in apprenticeship learning, no rewards are known at all. GPIRL therefore needs to modify GP regression in order to incorporate a stochastic relationship between the executed actions and the unknown rewards (Levine et al., 2011). As in previous, probabilistic approach we assume that the expert choose an action a in state s with probability proportional to the exponential of $Q(s, a)$ (see Equation (3.8)).

GPIRL learns two things at the same time. First of all the true rewards $\boldsymbol{u}$ for a feature coordinates $\boldsymbol{X_u}$, where $\boldsymbol{X_u}$ is a matrix containing the feature values for some inducing points. These inducing points could for example be all states encountered during demonstration or a subset of these. Furthermore, GPIRL learns the hyperparameters of the GP kernel $\boldsymbol{\theta}$. Finding $\boldsymbol{u}$ and $\boldsymbol{\theta}$ can be achieved by maximizing their joint posterior:

$$P(\boldsymbol{u}, \theta|\mathcal{D}, \boldsymbol{X_u}) \propto P(\mathcal{D}, \boldsymbol{u}, \theta|\boldsymbol{X_u}) = \left[\int_{\boldsymbol{R}} \underbrace{P(\mathcal{D}|\boldsymbol{R})}_{\text{IRL Term}} \underbrace{P(\boldsymbol{R}|\boldsymbol{u}, \boldsymbol{X_u}, \theta)}_{\text{GP Posterior}} d\boldsymbol{R}\right] \underbrace{P(\boldsymbol{u}, \theta|\boldsymbol{X_u})}_{\text{GP prior}}, \tag{3.16}$$

where $\boldsymbol{R} \in \mathbb{R}^{|\mathcal{S}|}$ denotes a vector containing the reward for all states. Since the integral in Equation (3.16) is impossible to compute in closed form (Levine et al., 2011), GPIRL uses an approximation related to sparse GP regression (Rasmussen, 2004) which sets $\boldsymbol{R} = \boldsymbol{K}_{\boldsymbol{R},\boldsymbol{u}}^{\mathsf{T}} \boldsymbol{K}_{\boldsymbol{u},\boldsymbol{u}}^{-1} \boldsymbol{u}$, where $\boldsymbol{K}_{\boldsymbol{R},\boldsymbol{u}}$ and $\boldsymbol{K}_{\boldsymbol{u},\boldsymbol{u}}$ are the covariance matrices as induced by the kernel function of the GP. Using this approximation results in the following optimisation problem for finding $\boldsymbol{u}$ and $\boldsymbol{\theta}$ (Levine et al., 2011):

$$\max_{\boldsymbol{u},\boldsymbol{\theta}} \log P(\mathcal{D}, \boldsymbol{u}, \boldsymbol{\theta}|\boldsymbol{X_u}) = \log P(\mathcal{D}|\boldsymbol{R} = \boldsymbol{K}_{\boldsymbol{R},\boldsymbol{u}}^{\mathsf{T}} \boldsymbol{K}_{\boldsymbol{u},\boldsymbol{u}}^{-1} \boldsymbol{u}) + \log P(\boldsymbol{u}, \boldsymbol{\theta}|\boldsymbol{X_u}). \tag{3.17}$$

Having found $\boldsymbol{u}$ and $\boldsymbol{\theta}$ by solving Equation (3.17), the approximated reward $\boldsymbol{R} = \boldsymbol{K}_{\boldsymbol{R},\boldsymbol{u}}^{\mathsf{T}} \boldsymbol{K}_{\boldsymbol{u},\boldsymbol{u}}^{-1} \boldsymbol{u}$ is the solution for the inverse reinforcement learning problem.

Instead of Gaussian processes, non-linear inverse reinforcement learning can also

be achieved using neural networks. Deep inverse reinforcement learning (DeepIRL) (Wulfmeier, Ondruska & Posner, 2015) adopts this idea by approximating the reward function as $R(s) = f(s, \theta)$, where f is a neural network and θ are its parameters. Their experiments show that DeepIRL performs slightly worse than GPIRL and is more sample inefficient. However, neural networks are more efficient in computing predictions for unseen states than Gaussian processes ($\mathcal{O}(1)$ instead of $\mathcal{O}(n^3)$, where n is the number of inducing points) (Wulfmeier et al., 2015).

3.3 Interactive Learning

Instead of providing the agent with a set of trajectories generated by an expert a priori, we can also model the apprenticeship learning problem interactively. Hereby, we essentially allow the agent to ask the expert for either advice or feedback. In the following we will present two different approaches which leverage this possibility of interaction. The first approach will allow interactions in form of queries from the agent to the expert for advice which action should be taken in a given state. The second approach instead lets the agent sample two trajectories and then asks the expert which trajectory should be preferred to the other.

3.3.1 Active Inverse Reinforcement Learning

When asking experts to demonstrate a task, the sampled trajectories often contain redundant information. In many settings, the start state and therefore most likely the expert's first few actions are identical over multiple runs. Depending on how strongly stochastic the environment is, the expert's path through the state space may then be fairly similar over several trajectories. Using these demonstrations for apprenticeship learning may then result in inefficiency due to the redundant information and furthermore in high uncertainty in rarely encountered states. Considering the example of teaching an agent how to drive autonomously by demonstrating the task will contain many state action pairs in which a skilled driver takes no or only insignificant actions to change the direction of the car, e.g. when driving on a straight road. We therefore have many state-action pairs representing almost identical situations. Furthermore, there are many situations which are rarely encountered, most importantly very critical situations such as emergency brakes or even collisions. An Apprenticeship Learner would therefore be fairly uncertain what to do in these situations.

(Lopes, Melo & Montesano, 2009) present an active learning approach with uncertainty sampling to deal with this problem. Their main idea is to first present the agent with an initial trajectory and afterwards allowing it to query the expert for advice for the state which the agent is most uncertain what action should be executed. This is similar to teaching a student: first presenting him some examples of how to solve a task and then allowing him to ask for help if he is stuck.

More formally, the active learning algorithm starts from an initial demonstration and then uses Bayesian Inverse Reinforcement Learning (see section 3.2.1) to estimate the reward posterior $P(\boldsymbol{R}|\mathcal{D})$. This induces the distribution over $P(\pi(s,a)|\mathcal{D})$. In order to compute a measure of uncertainty for a state s, we use the entropy $H(s)$ of state s. In order to use the more appealing (Lopes et al., 2009) discrete entropy instead of differential entropy, $P(\pi(s,a)|\mathcal{D})$ needs to be discretised, which can be done by partitioning $[0,1]$ into K subintervals $I_k = (\frac{k}{K}, \frac{k+1}{K}]$. Using this we can define:

$$\mu_{sa}(k) = P(\pi(s,a) \in I_k|\mathcal{D}) = \int_{I_k} P(\pi(s,a) = x|\mathcal{D})dx. \tag{3.18}$$

Given this and the definition of Shannon entropy, we can now compute the mean entropy $\bar{H}(s)$ as:

$$\bar{H}(s) = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} H(\mu(s,a)) = -\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\sum_{k=0}^{K-1} \mu_{sa}(k)\log\mu_{sa}(k). \tag{3.19}$$

Using this definition, the agent can now find the state where it is most uncertain, i.e. the state $s^* = \operatorname{argmax}_s \bar{H}(s)$. The complete method of using active learning for Inverse Reinforcement Learning is also shown in Algorithm 4. In order to perform active learning for inverse reinforcement learning, we require an expert who knows which action would be optimal or near-optimal in a state. In many settings, this can be however quite difficult. Considering the driving task again, presenting even a skilled driver with a snapshot of the situation (e.g. speed, distance to other obstacles, current steering angle, etc.), might now suffice for him to decide what to do, because humans need a sequence of information over time to judge a situation. And furthermore, humans may not be able to verbalise necessary actions since they usually drive intuitively. Asking an expert for naming or demonstrating an appropriate action for a given state can therefore be difficult. Instead we could present an expert with two state-sequences and ask him to judge which one performs a certain task better than the other. In the next section we will explain such an approach.

Algorithm 4 Active Learning for Inverse Reinforcement Learning

Require: Initial Demonstration $\mathcal{D}$
1: **repeat**
2: Estimate $P(\boldsymbol{R}|\mathcal{D})$ using Bayesian Inverse Reinforcement Learning
3: **for all** $s \in \mathcal{S}$ **do**
4: Compute $\bar{H}(s)$
5: **end for**
6: Query action for state $s^* = \operatorname{argmax}_s \bar{H}(s)$
7: Add sample to $\mathcal{D}$

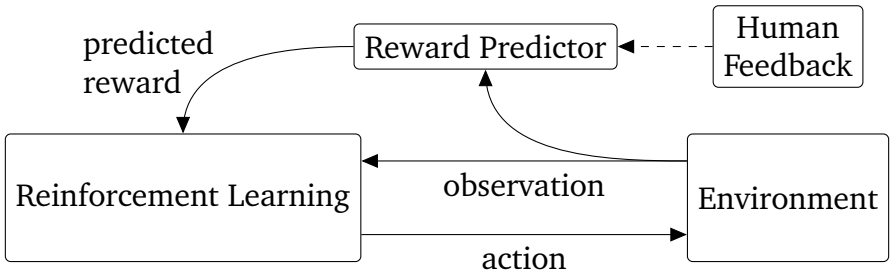

Figure 3.4: General structure of the apprenticeship learning from human preferences approach (Christiano et al., 2017).

3.3.2 Apprenticeship Learning from Human Preferences

All approaches presented so far assumed the availability of an expert who is able to demonstrate the task we want to learn. Unfortunately, there are tasks which humans are unable to demonstrate or only achieve poor performance. One such task is the control of a multi-joint robot, which would require a lot of training for a human to control it well. Nonetheless, humans are usually still able to judge whether a presented execution of a task is useful or not. Based on this idea, (Christiano et al., 2017) present an approach to apprenticeship learning in which humans do not need to demonstrate a task but instead only need to judge which of two task executions is better.

The general structure of (Christiano et al., 2017) is shown in Figure 3.4. The Reward Predictor and the Reinforcement Learning part are hereby implemented as neural networks which compute the reward estimate $\hat{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and the policy $\pi : \mathcal{S} \to \mathcal{A}$ respectively. Their algorithm works iteratively by first letting the current policy interact with the environment to generate a set of trajectories $\mathcal{D} = \{\zeta_1, \ldots, \zeta_N\}$. The reward predictor is meanwhile fixed, such that the policy network can be updated using a classical reinforcement learning algorithm. After this, two random segments σ_1, σ_2 are selected from $\mathcal{D}$. These are then presented to a human, who has to decided which segment is better performing a task than the other or if they perform equally well. More formally, the human returns a distribution μ over $\{1, 2\}$ which is defined as follows:

$$\begin{aligned} \sigma_1 \succ \sigma_2 : &\quad \mu(1) = 1, \mu(2) = 0 \\ \sigma_2 \succ \sigma_1 : &\quad \mu(1) = 0, \mu(2) = 1 \\ \sigma_1 = \sigma_2 : &\quad \mu(1) = \mu(2) = 0.5 \end{aligned} \tag{3.20}$$

Using this preference, the reward predictor network can be updated as follows. The probability of σ_1 being preferred over σ_2 can be modelled using a distribution which exponentially depends on the current accumulated reward estimate:

$$P[\sigma_1 \succ \sigma_2] = \frac{\exp \sum_t \hat{r}(s_t^1, a_t^1)}{\exp \sum_t \hat{r}(s_t^1, a_t^1) + \exp \sum_{t'} \hat{r}(s_{t'}^2, a_{t'}^2)} \tag{3.21}$$

Which can then be used to compute a loss function for the reward estimate $\hat{r}$:

$$\texttt{loss}(\hat{r}) = -\sum_{(\sigma_1,\sigma_2,\mu)\in\mathcal{D}} \mu(1)\log P[\sigma_1 \succ \sigma_2] - \mu(2)\log P[\sigma_2 \succ \sigma_1] \tag{3.22}$$

(Christiano et al., 2017) performed several experiments using this approach. First, they showed that their method could compete with reinforcement learners trained using the true reward function on several classical reinforcement learning task and on Atari games. However more interestingly, they showed that their method could teach agents novel tasks which could hardly be described as a reward function or be demonstrated by a human. Such tasks include for example, teaching a hopper robot to backflip or teaching a half-cheetah robot to move on one leg (Christiano et al., 2017).

Chapter 4

Experiments

After presenting several apprenticeship learning techniques in the last chapter, we will now perform some experimental tests using these methods. We start by presenting a comparison of some imitation and intention learning algorithms on a grid world and afterwards we test an interactive apprenticeship learning appraoch and attempt to teach a cart pole how to balance a stick by giving feedback on some of his actions.

4.1 Gridworld

We start our experiments with a simple grid world as shown in Figure 4.1 (adapted from (Pieter Abbeel & Ng, 2004)). The 16 by 16 grid is divided into 64 macrocells of size 2 by 2. All four cells of a macrocell share the same reward. The features of a state $\phi(s)$ can be represented as a 64-dimensional vector whose i-th element is 1 if s belongs to macrocell i and 0 otherwise. Using this we can express the true reward function as $R^*(s) = \boldsymbol{w}^{*\mathsf{T}}\phi(s)$, where $\boldsymbol{w}_i^*$ denotes the reward of macrocell i. In this scenario, the agent is allowed to freely move through the grid by choosing to either walk up, down, left or right. However, with probability 0.3 the action fails and the agent moves in a random direction instead. Each scenario starts in a randomly chosen cell and ends after 20 time steps.
In our first experiment we compared Feature Matching (section 3.2.1), MLIRL (section 3.2.2) and supervised imitation learning (section 3.1.1). Instead of a human expert, we used value iteration with the true rewards to find the optimal policy and then extract trajectories from this agent to train our apprenticeship learners.

Figure 4.2 shows the results for this first experiment. It can be seen that all three methods come close to the expert's performance, however inverse reinforcement learning based approaches (Feature Matching and MLIRL) are able to achieve these results using way fewer demonstrations than behavioural cloning. Considering that the state space of our used grid world is fairly small (only 64 states), this implies that behavioural cloning is not sample-efficient in enough to be transferred to larger problems. The results furthermore indicate that MLIRL is worse than Feature Matching when dealing with small sample sizes but is able to outperform Feature Matching

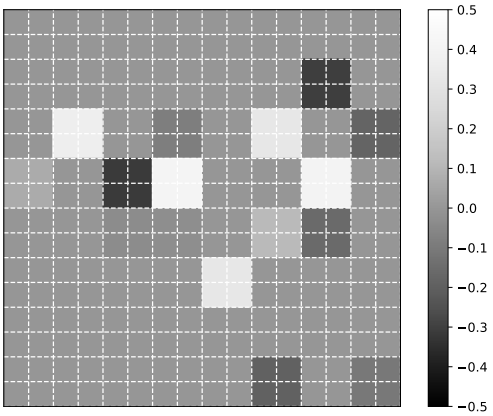

Figure 4.1: 16 by 16 gridworld consisting of 64 macrocells. Coloring represents the reward.

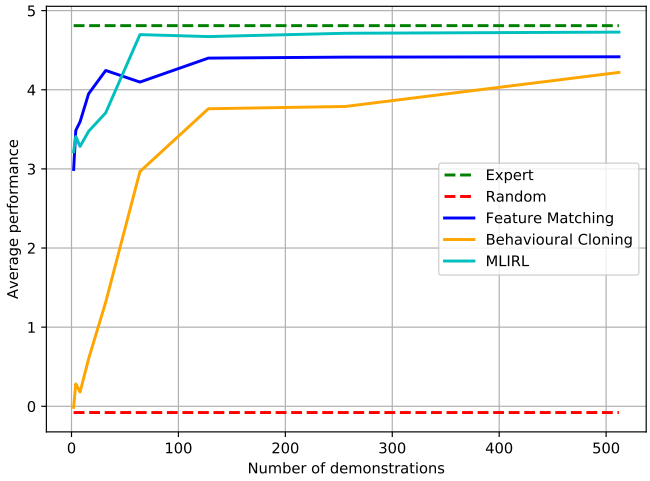

Figure 4.2: Experimental comparison of different apprenticeship learning approaches. Performance is displayed relatively to the expert policy (green) and a random policy (red). All results are averaged over 10 runs.

(and is almost levelled with the expert's performance) when more demonstrations are given.

We additionally visualised the recovered rewards of MLIRL in Figure 4.3. This shows that MLIRL is able to extract the general reward structure quite well even though not all rewards values could be exactly recovered.

4.2 Cart Pole

Our second experiment is concerned with an interactive apprenticeship learning task. More specifically, we are trying to use the apprenticeship learning from hu-

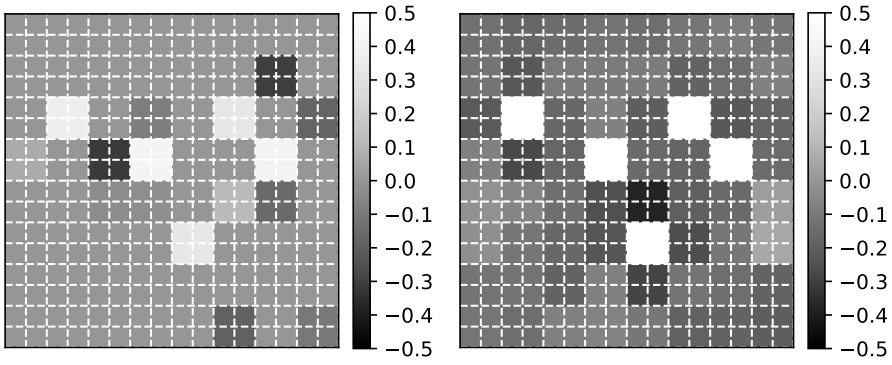

Figure 4.3: Recovered rewards using the MLIRL method. Left: Original rewards. Right: Rewards recovered by MLIRL using 64 trajectories.

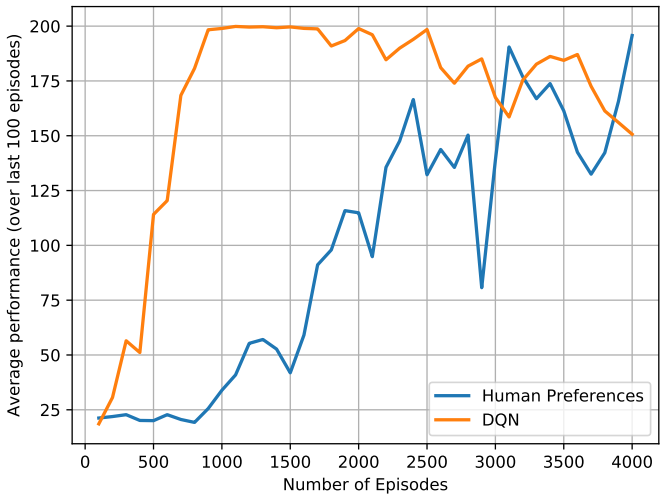

Figure 4.4: Comparison of the apprenticeship from human preferences approach versus a Deep Q-Network trained using the true environment rewards.

man preferences approach (see section 3.3.2 and (Christiano et al., 2017)) in order to teach an agent how to balance a cart pole. This experiment was first introduced by (Barto et al., 1983). For simplicity reasons, we again do not use a human expert but instead use an oracle (as proposed by (Christiano et al., 2017)) in order to provide preferences over two trajectories. This is done by evaluating the true reward of the environment for both trajectories and then telling the apprenticeship learner only which trajectory has performed better. Our implementation alternates between training a DQN for 100 episodes and sampling random state sequences from these episodes, judging them using the oracle and updating the reward predictor based on the oracle feedback. For comparison, we use the same DQN which has been used for the apprenticeship learning task, however with the true environment rewards. We implemented our algorithms using the environment form OpenAI Gym (Brockman et al., 2016).

The results of this experiment are shown in Figure 4.4. It is easy to see, that apprenticeship learning from human preferences took much longer to achieve the same performance. However, it also shows that the agent was indeed able to learn to execute the task really well, even though he only received feedback about some of his trajectories.

In their experiments, (Christiano et al., 2017) showed that apprenticeship learning from human preferences is able sometimes outperform reinforcement learners. This is however most likely only possible in more challenging environments than Cart Pole. Especially, in situations where rewards are very sparse and therefore rarely encountered, human feedback might be able to be more efficient than classical reinforcement learners. However, we have yet to formally evaluate this.

Chapter 5

Conclusion

Apprenticeship learning provides mechanism which enable us to teach an agent by demonstration or interaction. This addresses on of the key difficulties of reinforcement learning, namely that it is hard to encode a task or a goal within a reward function. Learning from observing and imitating is very common in human learning. Apprenticeship learning brings the field of artificial intelligence therefore another step closer to human intelligence.

The field of apprenticeship learning has come a long way. Its evolution started with imitation learning which blindly mimics the expert using supervised. This approach is fairly easy to implement, however it requires a good coverage of the state space in order to achieve reasonable results. Learning from demonstration then developed methods which do not try to imitate but to comprehend the reasons and motivations of the demonstrator in order to develop more general policies. This also enables us to handle problems such as suboptimal expert behaviour in a principled way.

In recent years however, interactive learning has raised more and more attention. Imitation and intention learning required the availability of an expert who performs the desired task quite well. This however restricts apprenticeship learning to applications which humans have already figured out. Some approaches from the interactive apprenticeship learning domain however are capable of learning without demonstrations but instead only require human feedback (see section 3.3.2). Judging whether a proposed behaviour is good or bad is way easier for humans than actually performing a task themselves. Future work might focus more on interactive or even symbiotic learning settings for machines and humans.

Bibliography

Abbeel, P. [P.], Dolgov, D., Ng, A. Y. & Thrun, S. (2008, September). Apprenticeship learning for motion planning with application to parking lot navigation. In *2008 ieee/rsj international conference on intelligent robots and systems* (pp. 1083–1090). doi:10.1109/IROS.2008.4651222

Abbeel, P. [Pieter] & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on machine learning*. ICML '04. Banff, Alberta, Canada: ACM. doi:10.1145/1015330.1015430

Babes, M., Marivate, V., Subramanian, K. & Littman, M. L. (2011). Apprenticeship learning about multiple intentions. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 897–904).

Bain, M. & Sammut, C. (1995). A framework for behavioural cloning. In *Machine intelligence 15*.

Bakker, P. & Kuniyoshi, Y. (1996). Robot see, robot do: an overview of robot imitation. In *Aisb96 workshop on learning in robots and animals* (pp. 3–11).

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5), 834–846.

Bloem, M. & Bambos, N. (2014). Infinite time horizon maximum causal entropy inverse reinforcement learning. In *Decision and control (cdc), 2014 ieee 53rd annual conference on* (pp. 4911–4916). IEEE.

Bogdanovic, M., Markovikj, D., Denil, M. & De Freitas, N. (2015). Deep apprenticeship learning for playing video games. In *Aaai workshop: learning for general competency in video games*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Choi, J. & Kim, K.-E. (2011). Map inference for bayesian inverse reinforcement learning. In *Advances in neural information processing systems* (pp. 1989–1997).

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S. & Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in neural information processing systems* (pp. 4302–4310).

Coates, A., Abbeel, P. & Ng, A. Y. (2009, July). Apprenticeship learning for helicopter control. *Commun. ACM*, *52*(7), 97–105. doi:10.1145/1538788.1538812

Freund, Y. & Schapire, R. E. (1999). Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, *29*(1-2), 79–103.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Hester, T. et al., Vecerik, M. et al., Pietquin, O. et al., Lanctot, M. et al., Schaul, T. et al., Piot, B. et al., ... Et al. et al. et al., Dulac-Arnold, G. et al. (2018). Deep q-learning from demonstrations. In *Proceedings of the conference on artificial intelligence (aaai)*.

Ho, J. & Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems* (pp. 4565–4573).

Huang, R. et al., Zhang, S. et al., Li, T. et al., He, R. et al. (2017). Beyond face rotation: global and local perception gan for photorealistic and identity preserving frontal view synthesis. *arXiv preprint arXiv:1704.04086*.

Kuefler, A., Morton, J., Wheeler, T. & Kochenderfer, M. (2017). Imitating driver behavior with generative adversarial networks. In *Intelligent vehicles symposium (iv), 2017 ieee* (pp. 204–211). IEEE.

Levine, S., Popovic, Z. & Koltun, V. (2010). Feature construction for inverse reinforcement learning. In *Advances in neural information processing systems* (pp. 1342–1350).

Levine, S., Popovic, Z. & Koltun, V. (2011). Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in neural information processing systems* (pp. 19–27).

Lopes, M., Melo, F. & Montesano, L. (2009). Active learning for reward estimation in inverse reinforcement learning. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 31–46). Springer.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V. et al., Kavukcuoglu, K. et al., Silver, D. et al., Rusu, A. A. et al., Veness, J. et al., Bellemare, M. G. et al., ... Et al. et al. et al., Ostrovski, G. et al. (2015).

Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Moore, A. W. (1990). Efficient memory-based learning for robot control.

Ng, A. Y. et al., Russell, S. J. et al. (2000). Algorithms for inverse reinforcement learning. In *Icml* (pp. 663–670).

Ramachandran, D. & Amir, E. (2007). Bayesian inverse reinforcement learning. *Urbana*, *51*(61801), 1–4.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In *Advanced lectures on machine learning* (pp. 63–71). Springer.

Russell, S. (1998). Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on computational learning theory* (pp. 101–103). ACM.

Sammut, C., Hurst, S., Kedzier, D. & Michie, D. (1992). Learning to fly. In *Machine learning proceedings 1992* (pp. 385–393). Elsevier.

Schaal, S. (1997). Learning from demonstration. In *Advances in neural information processing systems* (pp. 1040–1046).

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, *3*(6), 233–242.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).

Segre, A. & DeJong, G. (1985). Explanation-based manipulator learning: acquisition of planning ability through observation. In *Robotics and automation. proceedings. 1985 ieee international conference on* (Vol. 2, pp. 555–560). IEEE.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement learning: an introduction*. MIT press Cambridge.

Syed, U. & Schapire, R. E. (2008). A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems* (pp. 1449–1456).

Vroman, M. C. (2014). *Maximum likelihood inverse reinforcement learning*. Rutgers The State University of New Jersey-New Brunswick.

Wulfmeier, M., Ondruska, P. & Posner, I. (2015). Deep inverse reinforcement learning. *CoRR, abs/1507.04888*.

Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X. & Metaxas, D. (2017). Stackgan: text to photo-realistic image synthesis with stacked generative adversarial networks. In *Ieee int. conf. comput. vision (iccv)* (pp. 5907–5915).

Ziebart, B. D., Maas, A. L., Bagnell, J. A. & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Aaai* (Vol. 8, pp. 1433–1438). Chicago, IL, USA.