Inverse Optimal Control

Oleg Arenz
Technische Universität Darmstadt
o.arenz@gmx.de

Abstract

In Reinforcement Learning, an agent learns a policy that maximizes a given reward function. However, providing a reward function for a given learning task is often non trivial. Inverse Reinforcement Learning, which is sometimes also called Inverse Optimal Control, addresses this problem by learning the reward function from expert demonstrations. The aim of this paper is to give a brief introduction to Inverse Reinforcement Learning. It thereby focuses on two recent approaches which employ Gaussian Processes to learn nonlinear reward functions and maximum causal entropy to model the experts behavior. While the first approach is infeasible if the state space is too large, the second, more recent approach was successfully applied to high dimensional and continuous learning tasks.

1 Introduction

In Reinforcement Learning the actions of an agent produce rewards that serve as feedback to the agent, evaluating its performance in respect to a given goal. As the performance of an action depends on the state it was triggered, the reward depends on both: the action and the state. The learning task of the agent is to always choose an action that maximizes the expected accumulated future reward. Several algorithms have been successfully applied to solve the Reinforcement Learning problem - i.e. to learn a policy from a given reward function.
However, when applying Reinforcement Learning to a learning task, the reward function usually has to be hand-tuned. Therefore, obtaining a suitable reward function might emerge as a major challenge even if we assume that a human agent would naturally assign some form of reward to the available actions For example, a chess player might evaluate a certain chess position by extracting the important features of that position and weighting them up against each other. This evaluation could be represented by a linear reward function. However, when asked to formulate this reward function, the chess player would most probably fail for two reasons. Firstly, he would not be able to enumerate all possible features because he would only identify some of them while examining a concrete position. Second, a human chess player would assign fuzzy weights to the features and does not know their exact values.
Inverse Reinforcement Learning (IRL) - also referred to as Inverse Optimal Control - addresses the problem of finding a reward function by learning it from expert demonstrations. Providing some demonstrations is usually much easier than specifying the reward function directly. However, when deducing a reward function from a demonstration, it is not sufficient to choose any reward function that would make the demonstration optimal in respect to that function, because any constant reward function would adhere to this criterion. A constant reward function, however, would hardly pose the desired solution of Inverse Reinforcement Learning. Maximizing the accumulated differences of the rewards of the demonstrated actions and the second best actions is a possible approach to prevent such degenerated solutions [1]. Maximum Entropy IRL [2] is a more recent approach that assumes that the probability of a given demonstration is proportional to the exponential of the sum of rewards along that demonstration. Applying this model has the additional advantage that it assumes the expert demonstration to be only near optimal which is often the case with human demonstrations.
Additionally, the structure of the reward function has to be chosen. A linear reward function natu-

rally works best if the underlying true reward function is linear as well. However, if the underlying reward function is non-linear the accuracy of a linear approximation of that reward function might turn out to be insufficient to serve as a basis for Reinforcement Learning, especially if the correct features are not known. In these cases a non-linear structure of the reward function is usually required. The method of Gaussian Process Regression can be used to learn such nonlinear reward functions.
This paper serves as a short introduction to Inverse Reinforcement Learning and discusses recent approaches that combine Maximum Entropy IRL with non-linear reward functions. Section 2 provides some background information that is relevant to the subsequent sections. After giving an introduction to Reinforcement Learning and Inverse Reinforcement Learning both the principle of maximum entropy and the method of Gaussian Process Regression will be discussed in respect to IRL. Section 3 will cover GPIRL [3], an IRL approach that utilizes GP regression to learn a non-linear reward function while applying Maximum Entropy to model the experts behavior on a finite, discrete state and action space. Section 4 will discuss an algorithm that makes GPIRL feasible on a continuous state space by using a local approximation of the reward function and thereby assuming the expert demonstration only to be locally optimal instead of globally optimal. Finally, section 5 will recap both approaches and point out their differences and similarities.

2 Background

The following sections will assume a basic knowledge of forward and inverse Reinforcement Learning, Maximum Entropy IRL and Gaussian Process Regression. Therefore, this section provides a brief introduction to these basics.

2.1 Inverse Reinforcement Learning

A Markov decision process (MDP) is defined by a tupel $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathbf{r})$, where both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ can be either discrete or continuous. $\mathcal{T}_{s'}^{sa}$ is the probability to reach state $s' \in \mathcal{S}$ after choosing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. A MDP is deterministic, if the successor state is well-defined by the current state and action. $\gamma \in [0, 1)$ is the discount factor and $\mathbf{r}$ is the reward function.
Reinforcement Learning can be applied to deduce a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from the MDP. An optimal policy π^* maximizes the expected discounted sum of rewards $E[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_{s_t} | \pi^*]$.
The expected discounted sum of rewards by starting in state s and using policy π is given by the Value function [4]

$$V^{\pi}(s) = r(s) + \gamma \sum_{s'} \mathcal{T}_{s'}^{s\pi(s)} V^{\pi}(s'). \tag{1}$$

The expected discounted sum of rewards by starting in state s and choosing action a and then continue by using policy π is given by the Q-function [4]

$$Q^{\pi}(s, a) = r(s) + \gamma \sum_{s'} \mathcal{T}_{s'}^{sa} V^{\pi}(s'). \tag{2}$$

According to Bellman's principle of optimality [5] a policy π^* is optimal if for all $s \in \mathcal{S}$

$$\pi^*(s) = \arg\max_a Q^{\pi}(s, a). \tag{3}$$

Hence, the RL problem can be solved by learning $Q^{\pi^*}(s, a)$ by backward induction [5] (Value Iteration) and using a policy that always chooses an action that maximizes $Q^{\pi^*}(s, a)$.
In Inverse Reinforcement Learning [1], the reward function $\mathbf{r}$ of the MDP is unknown and has to be learned from expert demonstrations which are assumed either to be optimal or near optimal with respect to $\mathbf{r}$.

2.2 Maximum Entropy Inverse Reinforcement Learning

Not all reward functions that explain the expert behavior are equally meaningful. Therefore, MaxEnt uses a probabilistic model of the expert's behavior that can be used to deduce a non-ambiguous solution for the reward function without assuming the experts demonstration to be optimal. Maximum Entropy IRL [2] uses the policy which reproduces given state-action features and, at the same

time has the maximum entropy. Under this model, the probability of an action $\mathbf{u}$ in state $\mathbf{s}$ is proportional to the exponential of its Q-function,

$$P(a|s) = \frac{1}{Z} \exp\left(Q(s,a)\right), \tag{4}$$

where Q(s,a) can be computed for a specific reward function by using Value Iteration. The partition function $Z = \sum_a \exp\left(Q(s,a)\right)$ can be expressed as the exponential of the softened Value-function [6]

$$V^{\text{soft}}(s) = \underset{a}{\text{softmax}}\, Q(s,a) = \log \sum_a \exp\left(Q(s,a)\right). \tag{5}$$

Therefore, the probability $P(a|s)$ is the exponential of the difference between $Q(s,a)$ and $V^{\text{soft}}(s)$

$$P(a|s) = \exp\left(Q(s,a) - V^{\text{soft}}(s)\right), \tag{6}$$

where $Q(s,a) - V^{\text{soft}}(s)$ is called the advantage function. This enables the computation of $P(a|s)$ by using a variant of Value Iteration on the "softened" Value-function $V^{\text{soft}}(s)$. As the number of states grows exponentially with the dimensionality of the state space ("curse of dimensionality") this computation is infeasible on high dimensional state spaces.
If the MDP is deterministic, an expert demonstration is specified by a starting state $\mathbf{x}_0$ and a finite amount of actions $\mathbf{u} = (\mathbf{u}_1, ..., \mathbf{u}_T)^T$. The probability of a given Path $\mathbf{u}$ under the Maximum Entropy model can then be expressed in terms of the reward function r [2]:

$$P(\mathbf{u}|\mathbf{x}_0) = \frac{1}{Z} \exp\left(\sum_t r(\mathbf{x}_t, \mathbf{u}_t)\right), \tag{7}$$

where $\mathbf{x}_t$ is the state that is reached after choosing $\mathbf{u}_t$ in state $\mathbf{x}_{t-1}$. The partition function Z corresponds to the sum of accumulated rewards along every possible path and is therefore again infeasible to compute on high dimensional state spaces.

2.3 Gaussian Process Regression

A Gaussian process (GP) is defined as an infinite collection of random variables, for which any finite subset $\mathbf{x} = (x_1, ..., x_D)$ has a joint multivariate Gaussian distribution

$$p(\mathbf{x}|\mathbf{m}, \Sigma) = (2\pi)^{-D/2} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T \Sigma^{-1} (\mathbf{x}-\mathbf{m})\right), \tag{8}$$

where $\mathbf{m}$ is the vector of means and Σ is the covariance matrix. A GP can be seen as a probability distribution over functions, such that sampling a random variable x_i corresponds to sampling a function and immediately evaluating it at position x_i. As a Gaussian process has an infinite amount of random variables, the means are specified by $m(\mathbf{x})$ and the entries of the covariance matrix are specified by the kernel function $k(\mathbf{x}, \mathbf{x}')$. A Gaussian Process can then be written as

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{9}$$

Gaussian Process Regression utilizes a GP to predict the values of an unknown function without committing to a specific structure of that function. For that purpose, the GP is first trained with a set of observations $\{(\mathbf{x}_0, f_0), ..., (\mathbf{x}_n, f_n)\}$ and then utilized to predict the function outputs $\mathbf{f}_*$ for the test points $\mathbf{x}_*$. The mean $m(\mathbf{x})$ can be usually set to zero, because the actual mean can be subtracted from the training data. The joint Gaussian probability distribution of training outputs and test outputs is then [7]

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_{x,x} & K_{x,x_*} \\ K_{x,x_*}^T & K_{x_*,x_*} \end{bmatrix}\right), \tag{10}$$

where the covariance matrix $K_{a,b}$ is composed of the entries $k(\mathbf{a}_i, \mathbf{b}_j)$. By conditioning the Gaussian prior distribution on the observation, the gaussian posterior $P(\mathbf{f}_*|\mathbf{x}_*, \mathbf{x}, \mathbf{f})$ can be shown to be [7]

$$\mathbf{f}_*|\mathbf{x}_*, \mathbf{x}, \mathbf{f} \sim \mathcal{N}(K_{x_*,x} K_{x,x}^{-1} \mathbf{f}, K_{x_*,x_*} - K_{x_*,x} K_{x,x}^{-1} K_{x,x_*}). \tag{11}$$

Therefore, by providing a kernel function and a set of observations the function can be approximated at any points $\mathbf{x}_*$ by the means $K_{x_*,x}K_{x,x}^{-1}\mathbf{f}$. A common choice for a kernel function is the parametrized squared exponential kernel function [3]

$$k(\mathbf{x}, \mathbf{x}') = \beta \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Lambda (\mathbf{x} - \mathbf{x}')\right). \tag{12}$$

This kernel is controlled by the hyperparameters $\theta = \{\beta, \Lambda\}$, where β is the overall variance and the diagonal matrix Λ specifies the weights of the individual features. The hyperparameters can be learned by maximizing the GP log marginal likelihood $\log P(\mathbf{f}, \theta|\mathbf{x})$. Assuming noise-free observations the GP log marginal likelihood is [7]

$$\log P(\mathbf{f}, \theta|\mathbf{x}) = -\frac{1}{2}\mathbf{f}^T K_{x,x}^{-1}\mathbf{f} - \frac{1}{2}\log|K_{x,x}| - \frac{n}{2}\log(2\pi). \tag{13}$$

3 GPIRL

GPIRL [3] applies Gaussian Process Regression to Inverse Reinforcement Learning in order to learn nonlinear reward functions. However, while Gaussian Process Regression requires a training set that consists of features and their corresponding rewards, the expert demonstration of an IRL task contains only features but no rewards. GPIRL addresses this problem by using Maximum Entropy IRL to model the likelihood of the demonstrations $\mathcal{D}$ under the rewards $\mathbf{r}$. GP regression is then used to learn the kernel hyperparameters θ as well as the noise-free underlying rewards $\mathbf{u}$ for some inducing points $\mathbf{X}_u$. The inducing points $\mathbf{X}_u$ are not restricted to those states visited during demonstrations, however, choosing all states in the state space for $\mathbf{X}_u$ would become computationally to costly. Therefore, $\mathbf{X}_u$ is chosen to contain all states visited in the example paths as well as some random states. The most likely values for $\mathbf{u}$ and θ for the expert demonstrations $\mathcal{D}$ and the inducing points $\mathbf{X}_u$ are then found by maximizing

$$P(\mathbf{u}, \theta|\mathcal{D}, \mathbf{X}_u) \propto P(\mathcal{D}, \mathbf{u}, \theta|\mathbf{X}_u) = \left[\int_r P(\mathcal{D}|\mathbf{r})P(\mathbf{r}|\mathbf{u}, \theta, \mathbf{X}_u)d\mathbf{r}\right] P(\mathbf{u}, \theta|\mathbf{X}_u). \tag{14}$$

Where $\mathcal{D} = \{\zeta_1, ..., \zeta_N\}$ and $\zeta_i = \{(s_{i,0}, a_{i,0}), ..., (s_{i,T}, a_{i,T})\}$ are the demonstrated paths.
The log likelihood of $\mathcal{D}$ under $\mathbf{r}$ can be expressed by using Equation (6)

$$\log P(\mathcal{D}|\mathbf{r}) = \sum_i \sum_t \log P(a_{i,t}|s_{i,t}) = \sum_i \sum_t \left(Q(s, a) - V^{\text{soft}}(s)\right). \tag{15}$$

According to Equation (11), the GP posterior $P(\mathbf{r}|\mathbf{u}, \theta, \mathbf{X}_u)$ has mean $K_{\mathbf{r},\mathbf{u}}K_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{u}$ and covariance $K_{\mathbf{r},\mathbf{r}} - K_{\mathbf{r},\mathbf{u}}K_{\mathbf{u},\mathbf{u}}^{-1}K_{\mathbf{r},\mathbf{u}}^T$ where the covariance matrices are computed with the kernel function given in Equation (12). The term $\log P(\mathbf{u}, \theta|\mathbf{X}_u)$ is the log prior probability of $\mathbf{u}$ and θ and similar to Equation (13). However a term for the prior probability of the hyperparameters has been added to penalize partial correlation of feature points

$$\log P(\mathbf{u}, \theta|\mathbf{X}_\mathbf{u}) = -\frac{1}{2}\mathbf{u}^T K_{u,u}^{-1}\mathbf{u} - \frac{1}{2}\log|K_{u,u}| - \frac{n}{2}\log(2\pi) + \log P(\theta). \tag{16}$$

As the computation of $P(\mathcal{D}|\mathbf{r})$ requires a forward RL solver to calculate the Value function $V^{\text{soft}}(s)$, the integral in Equation (14) can not be solved in closed form. In order to maximize $P(\mathbf{u}, \theta|\mathcal{D}, \mathbf{X}_u)$ the GP posterior $P(\mathbf{r}|\mathbf{u}, \theta, \mathbf{X}_u)$ is assumed to have zero variance. Under this approximation the integral disappears and the log likelihood can be maximized

$$\log P(\mathcal{D}, \mathbf{u}, \theta|\mathbf{X}_u) \approx \log P(\mathcal{D}|\mathbf{r} = K_{\mathbf{r},\mathbf{u}}K_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{u}) + \log P(\mathbf{u}, \theta|\mathbf{X}_u). \tag{17}$$

Equation (17) can be minimized by the gradient-based L-BFGS method. However, computing the partial derivative $\partial(\log P(\mathcal{D}|\mathbf{r}))/\partial\mathbf{r}$ involves computing the Value function $V^{\text{soft}}(s)$ for the current reward $\mathbf{r}$. This computation makes the application of GPIRL infeasible for continuous states or large dimensional state spaces.

3.1 Experiments

The performance of GPIRL was evaluated using two different experiments [3]. In both experiments the algorithm was compared to several previous, existing IRL methods, including the linear

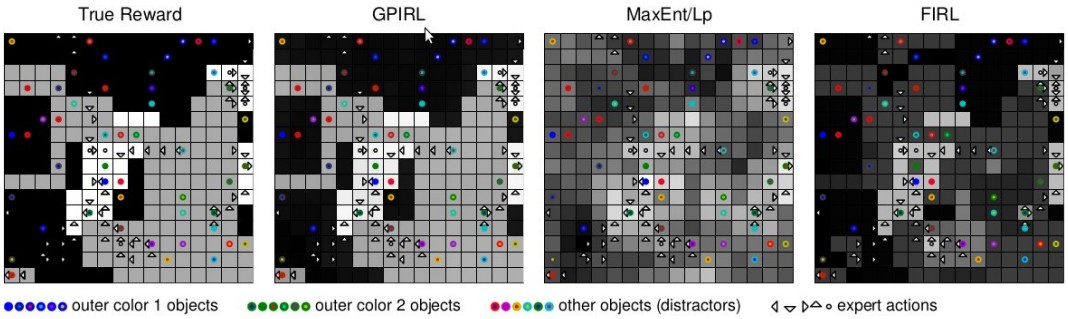

Figure 1: Comparison of the true reward and the rewards learned by the different algorithms in the grid world. Figure taken from "Nonlinear Inverse Reinforcement Learning with Gaussian Processes", Levine et.al 2011 [3].

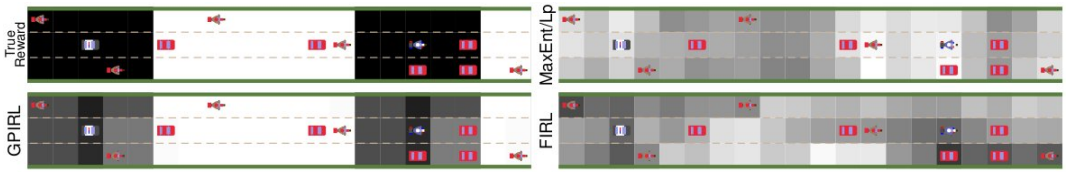

Figure 2: Comparison of the true reward and the rewards learned by the different algorithms in the highway driving experiment. Figure taken from "Nonlinear Inverse Reinforcement Learning with Gaussian Processes", Levine et.al 2011 [3].

Maximum Entropy IRL algorithm [2] and FIRL [8]. The algorithms were compared by using the "expected value difference", which is the difference of the expected sum of discounted rewards of the true policy and the optimal deterministic policy under the learned rewards. The learned reward functions were evaluated on the environments where they were learned as well as on four additional "transfer environments".

The first experiment was performed on an $N \times N$ grid world that was populated by randomly placed objects. Each object was assigned one of C inner colors and one of C outer colors. Five actions were available per state, one action per each direction and one action for staying in place. Each action has a 30% chance that the agent moves to a different random direction. If the agent is within three cells of outer color 1 and within three cells of outer color 2 the true reward is positive. If the agent is within three cells of outer color 1 but not within three cells of outer color 2 the true reward is negative. If the agent is not within three cells of outer color 1 the true rewards is zero. The inner colors and all colors but 1 and 2 do not influence the true rewards and therefore serve as distractors. Suboptimal optimal human demonstrations were provided as a basis for the IRL algorithms. It was shown that GPIRL outperforms all other algorithm in the first experiment on the training environments as well as on the transfer environments. The diagram in Figure 1 is taken from [3] and shows that GPIRL was the only algorithm able to learn the true nonlinear reward function.

The second experiment was conducted on a simple highway driving simulator. Besides the vehicle controlled by the agent there are two kinds of vehicles, police vehicles and civilian vehicles, that all move at constant speed. The speed of the agent's vehicle can be chosen to be either one times, two times, three times or four times the speed of the traffic. Additionally, the agent can switch lanes. The demonstrations were provided partially by a stochastic policy and partially by humans. In both cases, the aspired policy was to drive as fast as possible if no police vehicle is within two car-lengths of the agent and to drive no more than double the speed of traffic if a police vehicle is within that range. In the highway driving experiment GPIRL performed only slightly better than its competitors on the training environment, but achieved larger improvement on the transfer environments. This result indicates that GPIRL was better at learning the true underlying nonlinear reward function allowing the algorithm to generalize correctly. The learned reward functions are shown in Figure 2 taken from [3].

Both experiments show that GPIRL can outperform prior methods if indeed the true reward function is nonlinear and the demonstrations are suboptimal.

4 Continuous Inverse Reinforcement Learning

Although GPIRL was shown to be capable of learning a nonlinear reward function, both experiments had a limited action and state space. Applying GPIRL to continuous states and actions is difficult, because computing the partition function of the Maximum Entropy model, $V^{\text{soft}}(s)$, requires a variant of Value Iteration and therefore becomes infeasible. This section will cover a recent approach [9] that addresses this problem by using a local approximation of the partition function. By using a local approximation, the algorithm assumes the expert to choose its actions based on local optimization instead of global planning. The assumption of local optimal examples is strictly less restrictive than the assumption of global optimality. Therefore, the assumption of local optimality is still valid if the expert demonstrations are actually globally optimal, but the assumption of global optimality is invalid if the expert demonstrations are actually only locally optimal. In both cases, the more restrictive assumption of global optimality leads to more constraints on the reward function. If the expert demonstrations are globally optimal, these additional constraints can be utilized to construct a more accurate reward function. However, if the expert demonstrations are locally optimal, the additional constraints are based on a false assumption and therefore misleading. Hence, the local approximation does not only scale better with the dimensionality, but interestingly also allows the algorithm to learn from local optimal demonstrations.

The algorithm assumes a deterministic MDP, allowing to express the Maximum Entropy action probability according to Equation (7) which can be also be written as

$$P(\mathbf{u}|\mathbf{x_0}) = e^{r(\mathbf{u})} \left[\int e^{r(\tilde{\mathbf{u}})} d\tilde{\mathbf{u}} \right]^{-1}, \tag{18}$$

where $r(\mathbf{u}) = \sum_t r(\mathbf{x}_t, \mathbf{u}_t)$ is the reward for a whole trajectory. As the MDP is deterministic, a trajectory starting at state $\mathbf{x}_0$ is uniquely described by the action $\mathbf{u}$ along that trajectory. Equation (18) is approximated by using a second order Taylor expansion of r around $\mathbf{u}$

$$r(\tilde{\mathbf{u}}) \approx r(\mathbf{u}) + (\tilde{\mathbf{u}} - \mathbf{u})^T \mathbf{g} + \frac{1}{2}(\tilde{\mathbf{u}} - \mathbf{u})^T \mathbf{H} (\tilde{\mathbf{u}} - \mathbf{u}), \tag{19}$$

where $\mathbf{g} = \frac{\partial r}{\partial \mathbf{u}}$ is the gradient and $\mathbf{H} = \frac{\partial^2 r}{\partial \mathbf{u}^2}$ is the Hessian. The Maximum Entropy action probability can now be approximated by combining Equation (18) and Equation (19)

$$P(\mathbf{u}|\mathbf{x}_0) \approx e^{r(\mathbf{u})} \left[\int e^{r(\mathbf{u}) + (\tilde{\mathbf{u}}-\mathbf{u})^T \mathbf{g} + \frac{1}{2}(\tilde{\mathbf{u}}-\mathbf{u})^T \mathbf{H} (\tilde{\mathbf{u}}-\mathbf{u})} d\tilde{\mathbf{u}} \right]^{-1} \tag{20}$$

$$= e^{-\frac{1}{2}\mathbf{g}^T \mathbf{H}^{-1} \mathbf{g}} \left[\int e^{\frac{1}{2}(\mathbf{H}(\tilde{\mathbf{u}}-\mathbf{u})+\mathbf{g})^T \mathbf{H}^{-1} (\mathbf{H}(\tilde{\mathbf{u}}-\mathbf{u})+\mathbf{g})} d\tilde{\mathbf{u}} \right]^{-1} \tag{21}$$

$$= e^{\frac{1}{2}\mathbf{g}^T \mathbf{H}^{-1} \mathbf{g}} |-\mathbf{H}|^{\frac{1}{2}} (2\pi)^{-\frac{d_\mathbf{u}}{2}}. \tag{22}$$

Note that the integral in Equation (21) corresponds to an integral over an un-normalized multivariate Gaussian distribution and, consequently corresponds to the reciprocal of the normalization factor $|-\mathbf{H}|^{\frac{1}{2}} (2\pi)^{-\frac{d_\mathbf{u}}{2}}$.

The approximate log likelihood of Equation (18) is now given by

$$\mathcal{L} = \frac{1}{2}\mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} + \frac{1}{2} \log |-\mathbf{H}| - \frac{d_\mathbf{u}}{2} \log 2\pi \tag{23}$$

The computation of Equation (23) is dominated by $\mathbf{h} = \mathbf{H}^{-1}\mathbf{g}$. By pretending $\frac{\partial^2 \mathbf{x}}{\partial \mathbf{u}^2} = 0$, $\mathbf{H}$ and $\mathbf{g}$ can be written entirely in terms of block diagonal or block triangular matrices which can be exploited when computing $\mathbf{h}$. The corresponding algorithm can be found in [9].

Gradient-based optimization of the reward function parameters θ can be used to optimize Equation (23) using the gradient

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} = &\sum_{t,i} \frac{\partial \tilde{\mathbf{g}}_{ti}}{\partial \theta} \mathbf{h}_{ti} + \sum_{t,i} \frac{\partial \hat{\mathbf{g}}_{ti}}{\partial \theta} [\mathbf{J}^T \mathbf{h}]_{ti} + \frac{1}{2} \sum_{t,i,j} \frac{\partial \tilde{\mathbf{H}}_{tij}}{\partial \theta} \left(\left[\mathbf{H}^{-1}\right]_{ttij} - \mathbf{h}_{ti}\mathbf{h}_{tj} \right) \\ &+ \frac{1}{2} \sum_{t,i,j} \frac{\partial \hat{\mathbf{H}}_{tij}}{\partial \theta} \left(\left[\mathbf{J}^T \mathbf{H}^{-1} \mathbf{J}\right]_{ttij} - \left[\mathbf{J}^T \mathbf{h}\right]_{ti} \left[\mathbf{J}^T \mathbf{h}\right]_{tj} \right), \end{aligned} \tag{24}$$

where $\tilde{\mathbf{g}} = \partial r/\partial \mathbf{u}$, $\hat{\mathbf{g}} = \partial r/\partial \mathbf{x}$, $\mathbf{J} = \partial \mathbf{x}^T/\partial \mathbf{u}$, $\tilde{\mathbf{H}} = \partial^2 r/\partial \mathbf{u}^2$, $\hat{\mathbf{H}} = \partial^2 r/\partial \mathbf{x}^2$ and $\left[\mathbf{H}^{-1}\right]_{ttij}$ denotes the ij^{th} entry of block tt. The diagonal blocks of $\mathbf{H}^{-1}$ and $\mathbf{J}^T\mathbf{H}^{-1}\mathbf{J}$ can be obtained while solving $\mathbf{h} = \mathbf{H}^{-1}\mathbf{g}$ [9].
Therefore, the reward function can be obtained by computing the gradients $\hat{\mathbf{g}}$, $\tilde{\mathbf{g}}$, $\hat{\mathbf{H}}$ and $\tilde{\mathbf{H}}$ for a given structure of r, then computing the gradients with respect to the parametrization θ and then using Equation (24) for gradient-based optimization. This procedure was shown for linear reward functions as well as for Gaussian processes [9].

4.1 Linear Reward Functions

Linear reward functions express the reward as a weighted sum of features $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$. The weight vector θ is the parameter that is to be learned so that $r(\mathbf{x}_t, \mathbf{u}_t) = \theta^T \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$. The gradients and Hessian $\hat{\mathbf{g}}$, $\tilde{\mathbf{g}}$, $\hat{\mathbf{H}}$ and $\tilde{\mathbf{H}}$ can be expressed by the weighted sum of the gradients of the individual features f_i, e.g., $\tilde{\mathbf{g}} = \frac{\partial r}{\partial u} = \sum_i \frac{\partial f_i}{\partial u}\theta_i$.
The logarithm of the negative Hessian in Equation (23) is undefined if the Hessian is not positive definite. In order to force the Hessian to be positive definite, a dummy feature is added so that it has a zero gradient and a Hessian equal to the negative identity matrix. The weight on that feature is set high enough to ensure that the Hessian is positive definite and gradually reduced during the optimization until it becomes zero and does not effect the solution.

4.2 Nonlinear Reward Functions

The approximation can also be used to learn a nonlinear reward function such as a Gaussian process. The approach is similar to the one discussed in Section 3, however, the IOC likelihood $P(\mathcal{D}|\mathbf{r})$ in Equation (17) is replaced by the approximate log likelihood of Equation (23) which again is maximized using the gradient calculated with Equation (24). Therefore, the gradients and Hessians $\hat{\mathbf{g}}$, $\tilde{\mathbf{g}}$, $\hat{\mathbf{H}}$ and $\tilde{\mathbf{H}}$ of the reward function have to be obtained.

The reward function $r(\mathbf{x}_t, \mathbf{u}_t)$ is given by the GP posterior mean and augmented with an additional set of linear features $\mathbf{f}_l$

$$r(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{k}_t \mathbf{K}^{-1}\mathbf{y} + \theta^T \mathbf{f}_l(\mathbf{x}_t, \mathbf{u}_t), \tag{25}$$

where $\mathbf{K}$ is the covariance matrix of the inducing (feature) points, and $\mathbf{k}_t$ is the covariance vector of the feature point $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ with respect to the inducing points.

The gradients and Hessians of r are then

$$\hat{\mathbf{g}}_t = \frac{\partial r(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t} = \frac{\partial \mathbf{k}_t}{\partial \mathbf{x}_t}\mathbf{K}^{-1}\mathbf{y} + \sum_i \frac{\partial \mathbf{f}_{l,i}(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t}\theta_i, \tag{26}$$

$$\hat{\mathbf{H}}_t = \frac{\partial^2 \mathbf{k}_t}{\partial \mathbf{x}_t^2}\mathbf{K}^{-1}\mathbf{y} + \sum_i \frac{\partial^2 \mathbf{f}_{l,i}(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t^2}\theta_i. \tag{27}$$

The gradients and Hessians of $\mathbf{k}_t$ are

$$\frac{\partial \mathbf{k}_t}{\partial \mathbf{x}_t} = \sum_k \left(\frac{\partial \mathbf{k}_t}{\partial \mathbf{f}_k(\mathbf{x}_t, \mathbf{u}_t)} \frac{\partial \mathbf{f}_k(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t} \right), \tag{28}$$

$$\begin{aligned} \frac{\partial^2 \mathbf{k}_t}{\partial \mathbf{x}_t^2} &= \sum_k \left(\frac{\partial \mathbf{k}_t}{\partial \mathbf{f}_k(\mathbf{x}_t, \mathbf{u}_t)} \frac{\partial^2 \mathbf{f}_k(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t^2} \right) \\ &+ \sum_{k_1, k_2} \left(\frac{\partial^2 \mathbf{k}_t}{\partial \mathbf{f}_{k_1}(\mathbf{x}_t, \mathbf{u}_t) \partial \mathbf{f}_{k_2}(\mathbf{x}_t, \mathbf{u}_t)} \frac{\partial \mathbf{f}_{k_1}(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t} \frac{\partial \mathbf{f}_{k_2}(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t} \right) \end{aligned} \tag{29}$$

The gradient and Hessian of the reward with respect to the actions are given analogously.

4.3 Experiments

The performance of both variants of the algorithm were evaluated on three different experiments [9] and compared with MaxEnt [2] and OptV [10]. In the first task the agent had to take steps on a

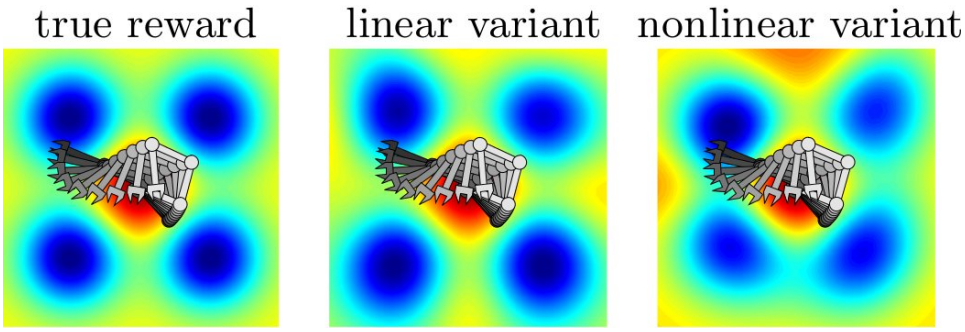

Figure 3: Comparison of the true reward and the rewards learned by the linear and nonlinear variant of the algorithm for the robot control experiment. Diagram taken from "Continuous Inverse Optimal Control with Locally Optimal Examples", Levine et.al 2012 [9].

plane. The reward function on that plane had its peak in the center, surrounded by four pits. The reward function was learned from either locally optimal or globally optimal examples. The learned reward function was evaluated on 32 random initial states. For that purpose, the optimal path with respect to the true reward function and the optimal path in respect to the learned reward function were computed. The reward loss was computed by subtracting the true reward gained along the learned path from the true reward gained along the optimal path. In this experiment both variants of the algorithm converged to the true policy, even when they were only presented with locally optimal examples. MaxEnt and OptV were not able to converge to the true policy from locally optimal examples.
In the second experiment, the agent had to control a simulated n-link robot arm on the same plane as in the first experiment. The robot arm was controlled by setting the continuous torque for each link and the reward was assigned according to the position of the end-effector. Additionally, the reward was reduced proportional to the sum of the squared torque. The states were defined by the speeds and angles of the robot link, resulting in a state space with $2n$ dimensions. The reward functions were learned from globally optimal examples for the 2-link robot arm and from locally optimal examples for higher dimensional tasks. Again, the reward functions were compared using the reward loss. The robot experiment tested the performance of the algorithms in respect to the task dimensionality by running the experiment for several numbers of robot arm links. Both algorithms scaled gracefully with the dimensionality and were able to solve the learning task for up to eight links. The reward loss was close to zero for all tested dimensions. The learned reward functions are illustrated in Figure 3, taken from [9]. MaxEnt and OptV could not compete, because their processing time scaled exponentially with the dimensionality.
In the third experiment, the agent had to control a simulated vehicle. Unlike the driving simulator in Section 3.1, this time the agent had direct continuous control of the gas, breaks and steering wheel. The states were continuous as well, including the position, orientation and linear and angular velocities. This experiment was only conducted by the nonlinear algorithm. The task of the experiment was to learn three different driving style from sixteen human demonstrations at a time. The aggressive driving style tries to cut off other cars, the evasive driving style tries to keep a safe distance to other cars and the tailgating style tries to follow the other cars closely. The algorithm did successfully copy all three driving styles. The learned reward functions are shown in Figure 4.

5 Conclusion

By combining Reinforcement Learning and Inverse Reinforcement Learning an agent is able to generalize expert demonstrations. This paper discussed two similar approaches in order to investigate the state-of-the-art of Inverse Reinforcement Learning. In IRL the structure of the learned reward function must be suitable to express the true reward. As the structure of the true reward is usually not known, both depicted approaches are capable of learning the structure online. GPIRL introduced the idea of using a Gaussian Process to learn the reward function. However, unlike to prior applications of GP regression, in IRL the target values of the inducing points are unknown. This dilemma was solved by using Maximum Entropy IRL to identify the maximum likelihood target values in

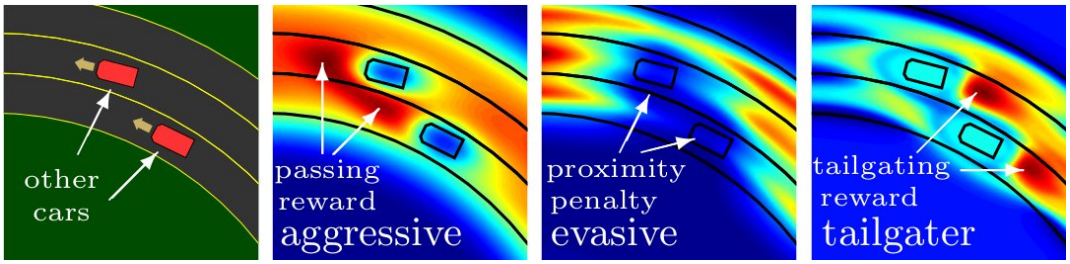

Figure 4: Reward functions learned from the different demonstrated styles. Diagram taken from "Continuous Inverse Optimal Control with Locally Optimal Examples", Levine et.al 2012 [9].

respect to the observed behavior. GPIRL did indeed outperform prior IRL algorithms on tasks with nonlinear reward functions, but the algorithm was only applicable for low dimensional state spaces, because computing the partition function was too costly. The second approached solved this limitation by using a second order Taylor expansion on the Maximum Entropy IRL distribution. This approximation does not directly depend on a specific structure of the reward function and therefore allows different variants of the algorithm. The linear variant used a weighted sum of features and the nonlinear variant was based on GPIRL. Due to the approximation, the algorithm was able to solve high-dimensional learning tasks, while assuming the examples to be only locally optimal. Interestingly, using a local approximation can be beneficial, because it allows the algorithm to learn from locally optimal examples. However, this algorithm is only applicable if the Markov Decision Process is deterministic.

References

[1] Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *Proceedings of the seventeenth International Conference on Machine Learning*, pages 663–670, 2000.

[2] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, pages 1433–1438, 2008.

[3] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in Neural Information Processing Systems*, 24, 2011.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[5] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[6] Brian D Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, University of Washington, 2010.

[7] Carl Edward Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[8] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 23, 2010.

[9] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. *Proceedings of the International Conference on Machine Learning (ICML2013)*, June 18 2012.

[10] Krishnamurthy Dvijotham and Emanuel Todorov. Inverse optimal control with linearly-solvable mdps. *Proceedings of the International Conference on Machine Learning*, 27:335–342, 2010.