Inverse Reinforcement Learning - A Brief Survey

Ajay Baldev Sailopal 2017MT10726
Sharut Gupta 2017MT60250
Sumanth Varambally 2017MT60855

1 Introduction

In the Markov Decision Processing (MDP) problem, we have to make a sequence of actions to optimize a reward function. Under the usual MDP setting, we assume that this reward function is known to us, as well as the transition probabilities and state dynamics wherein the optimal policy of actions can be exactly determined. Reinforcement Learning (RL) employs an agent which interacts with an environment to learn a policy to optimize an objective function. This is modelled using an MDP

However, we consider the reverse problem - reconstruct the reward function used when we can only observe the agent act out under the optimal policy. This is the Inverse Reinforcement Learning Problem (IRL). In practice, the reward function for an MPD is often manually tweaked - and it is not trivial to specify the exact reward function. This makes IRL a relevant problem as it broadens the applications of RL in general problems

While learning how to drive, the learner is often never given a specified reward function to drive. The learner observes a driver drive and 'learns by doing' and 'learns by observing'. This further motivates the relevance of IRL in applications such as self driving cars, etc. We model IRL problems as an MDP where the optimal policy is known but the reward function is unknown. In the car driving example, the policy is the set of actions which the driver makes while driving which are observed by the learner, and he must deduce the reward function on his own.

1.1 Relevance of IRL

Machine learning and Arificial Intelligence researchers have developed a significant interest for IRL

1.1.1 Better Reward Functions

As mentioned before in the car driving example, we are often not provided with a predetermined reward function. Moreover, finding a suitable reward function is not easy beyond intuition. This need to pre-specify a reward function limits the applications of RL. IRL hence broadens the horizons for RL by providing a way to get a numerically suitable reward function. Although we need to have access to an optimal policy, these demonstrations are usually readily available in practice

1.1.2 Better Transferability

A reward function can be thought of as a representation of the desire of an agent and can be transferred to another agent. If another agent has problem specifications that differ mildly from the first one, then the learned reward function provides a basis for the second one. However, this benefit of transferability is not enjoyed by the optimal policies. For example, if the second agent has even a larger state space with all other dynamics same as the first one, the optimal policy for the first one does not give any idea as to what the optimal policy for the second agent will be.

1.1.3 Applications

IRL can be used to imitate experts. Extending the self driving car example, we can make an agent learn the reward function of an expert using IRL, by simply allowing the agent to observe episodes of the expert's actions. Moreover, we can use IRL to make one agent mimic the actions of another agent in a similar way. We shall cover the applications of IRL in detail in section 4.

1.2 Purpose of this literature review

- To give a formal introduction to IRL and some applications
- To give an overview of some IRL algorithms along with analysis
- Perform simulations of our own to reflect their performance
- Study some challenges and potential for future work in this domain

1.3 Contents

Section 2 will cover some preliminaries and formalisations required for understanding IRL starting from MDPs and working our way up to formalising the IRL problem. Section 3 will cover IRL algorithms using Maximum margin methods, Entropy methods and Bayesian methods, while section 4 will cover some of the various applications of IRL algorithms and section 5 will look at some of the challenges posed in this problem setting, along with some variants to these algorithms.

2 Preliminaries and Definitions

In this section, we introduce terminologies and notations for a Markov Decision Process and accordingly define Inverse Reinforcement Learning.

2.1 Markov Decision Process

A Markov Decision Process (MDP) is represented as a tuple (S, A, T, R, γ) where:

- S denotes a finite collection of states or the state space

- A is the action space
- T is the state transition probability conditioned on the current state and action.
- R represents the reward function
- $\gamma \in [0, 1]$ represents the scalar discount factor responsible for weighting reward values accumulated over time

Using the above formulation, a Policy is defined as the function from the state space to the action space. Intuitively, it maps the current state to the next action (deterministic) or a collection of actions (stochastic). Given a policy π and a start state x_0, the value function $J_\pi(x_0)$ is given by

$$J_\pi(x_0) = E_{x,\pi(x)}\Big[\sum_{t=0}^{\infty} \gamma^t R(x_t, \pi(x_t))|x_0\Big] \tag{1}$$

Optimal policy π^* maximises the above value function i.e. $\pi^* = \max_\pi J_\pi(x_0)$.

For algorithmic simplicity, the reward function can be approximated as a linearly weighted sum of basis functions, also known as feature functions. Restricting to $S = \mathbb{R}^n$, feature functions are mappings from S to $\mathbb{R}$. Concretely,

$$R(x) = w_1\phi_1(x) + w_2\phi_2(x) + ... + w_d\phi_d(x) = w^T\phi(x) \tag{2}$$

where $w_i's$ are weights and $\phi's$ are fixed and bounded feature functions. In the above equation, reward function is considered as a function of state. However, other formulations with reward being a function of (x, a) with $x \in S, a \in A$ or (x, a, x') with $x' \in S$ denoting the next state, are also possible. Assuming linear approximation of the reward function and policy as an input i.e. $R(x) = w^T\phi(x)$, we have

$$\begin{aligned} J_\pi &= E_x\Big[\sum_{t=0}^{\infty} \gamma^t R(x_t)|\pi\Big] \\ &= E_x\Big[\sum_{t=0}^{\infty} \gamma^t w^T\phi(x)|\pi\Big] \\ &= w^T E_x\Big[\sum_{t=0}^{\infty} \gamma^t \phi(x)|\pi\Big] \\ &= w^T\mu(\pi) \end{aligned} \tag{3}$$

with $\mu(\pi) = E_x\Big[\sum_{t=0}^{\infty} \gamma^t\phi(x)|\pi\Big]$ being the expected discounted accumulated feature value vector or the feature expectations. Since, optimal policy satisfies $E_x\Big[\sum_{t=0}^{\infty} \gamma^t R^*(x_t)|\pi^*\Big] \geq E_x\Big[\sum_{t=0}^{\infty} \gamma^t R^*(x_t)|\pi\Big]$, and hence the problem reduces to find $w^* s.t. w^{*T}\mu(\pi^*) \geq w^{*T}\mu(\pi)$. Similarly, the reward function can also be approximated using a probability distribution over alternative rewards or using regression trees.

2.2 Reinforcement and Inverse Reinforcement Learning

Reinforcement learning provides a way to solve a given MDP. It uses the reward functions and transition probabilities to estimate an optimal policy.

Inverse Reinforcement Learning (IRL) on the contrary inverts the RL problem. Given a Markov Decision Process (MDP) without a reward function, a set of perfectly observable trajectories (or policy), IRL finds a reward function which best explains the desired trajectories (or policy). The input to IRL may include a policy or sequences of observed state-action pairs called trajectories. As shown in Fig 2, IRL inverts the process of RL which interacts with the environment and chooses the optimal policy based on the obtained rewards.

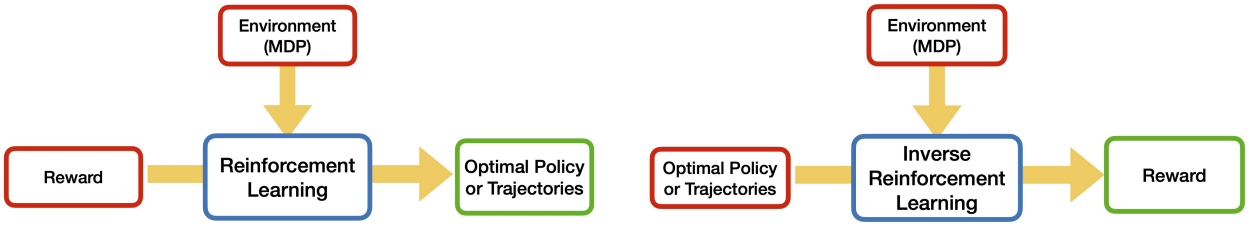


Figure 1: A schematic of (a) Reinforcement Learning and (b) Inverse Reinforcement Learning (IRL)

IRL closely resembles apprenticeship learning [1], which aims to derive a policy that is almost as good as the expert's trajectories under this reward function R^*. Consequently, apprenticeship learning has a weaker objective since it does not necessarily need to derive the true reward function. Concretely, given a set of perfectly observed expert trajectories $D_E = \{[(s_0^0, a_0^0), (s_1^0, a_1^0)..(s_{k_0}^0, a_{k_0}^0)], [(s_0^1, a_0^1), (s_1^1, a_1^1)..(s_{k_1}^1, a_{k_1}^1)]...[(s_0^m, a_0^m), (s_1^m, a_1^m)..(s_{k_m}^m, a_{k_m}^m)]\}$, IRL aims to find out R_E^* which best describes either the given expert trajectories or policy π_E.

3 Methods

3.1 Optimization of Margin

Algorithms under this category, aim to find the reward function which is better than other potential reward functions by a certain margin. Since most IRL formulations involve a small finite set of trajectories, a variety of reward functions (especially the degenerate functions) tend to best explain a given MDP. As a consequence, accurate inference is a challenge. Techniques of margin optimization help to solve this problem by choosing the solution with maximum margin from the other alternatives. Based on the choice of margin, previous works can be classified into two broad categories.

3.1.1 Margin using optimal policy

The simplest and the most intuitive choice of margin is to maximise the sum of differences between quality of optimal action and that of a best action [10]. This margin is given by

$$M = \sum_{x \in S} \left(Q^{\pi}(x, a') - \max_{a \in A/a'} Q^{\pi}(x, a) \right) \tag{4}$$

The above margin not only makes π optimal but also penalizes heavily on single step deviation from the optimal policy. Approximating the reward function as a a linear sum of basis functions as in Eq. 3, linear programming formulation is used to estimate the accurate reward function.

Radliff used maximum margin planning (MMP) to solve the framework of imitation learning by removing the uncertainty in obtained reward function [9]. Using linear hypothesis of reward and subgradient method of convex optimization, MMP finds the accurate reward function.

The assumption of approximating reward function as a weighted sum of feature functions is not feasible for practical problems since the quality of the learned policies can be greatly jeopardized with the error of value estimation. Radliff extended this study to learn non-linear reward function using LEArning and SeaRCH algorithms or LEARCH [8]. Using the same margin, LEARCH found accurate functions in legged locomotion, grasp planning, and autonomous outdoor unstructured navigation.

3.1.2 Margin using feature expectations

Given a set of expert trajectories, empirical estimate of expert's feature expectations (μ_E is given by

$$\mu_E = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{\infty} \gamma^t \phi(x^{(i)}) \tag{5}$$

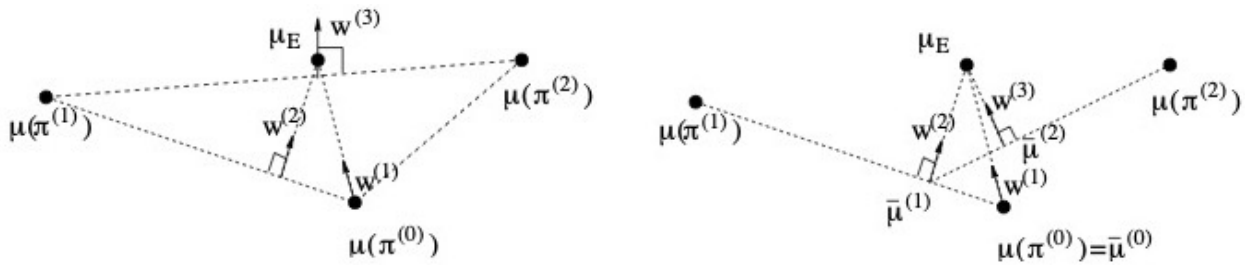
Assuming weighted linear sum approximation of reward function (Eq. 3), this subset of algorithms uses difference between estimate of feature expectations of expert trajectories and the learner as its margin. Mathematically,

$$M = |\mu(\pi^j) - \mu_E| \tag{6}$$

Abbeel [1] introduces two different algorithms namely Max-Margin method and the Projection method to perform IRL using expert trajectories. While the former needs quadratic programming (QP) or Support Vector Machine (SVM) to solve the problem, the later computes orthogonal projection of μ_E onto the line passing through last two iterates to avoid the need of a QP solver.

3.2 Entropy Based Methods

Intuitively, entropy can be thought of as the measure of randomness present in a set of data This can also be related with the amount of information present in this particular data. We use this to motivate our next approach of IRL algorithms, wherein we use entropy as a measure of accuracy or suitability of the reward function.



(a) (b)

Figure 2: A schematic of three iterations for (a) Max-margin method (b) projection algorithm.

3.2.1 Maximum Entropy Methods

One interesting way to approach IRL algorithms is by using Maximum Entropy methods, which posits that the policy which best represents the demonstrated behaviour is the one with the highest entropy, as long as the reward value for the actions matches that of the demonstrated actions. The second constraint can be ensured by constraining the feature counts of the learnt policy to match with that of the demonstrations, that is:

$$\sum_{\tau \in \mathcal{T}} \mathbf{Pr}(\tau|\pi, T) f_i^T = \hat{f}_i \ \ \forall i \in 1, 2, ..., k \tag{7}$$

where $\hat{f}_i$ denotes the sample expectation of the feature i from the demonstration. Another advantage of maximum entropy IRL is that it prevents issues with label bias, that is, the state space having more branches will be less likely, while areas having fewer branches will be given higher probability The problem is set up as follows - The agent is optimizing a reward function which is a linear combination of the features of each state f_s in the path τ to a state reward value, which is hence parameterised by a vector θ and is given by :

$$\text{Reward}(f_\tau) = \theta^T f_\tau = \sum_{s_j \in \tau} \theta^T f(s_j) \tag{8}$$

The expected empirical feature counts for m demonstrations is given by $\tilde{f} = \frac{1}{m} (\sum_i f_{\tilde{\tau}_i})$ The approach of Ziebart et al (2008) [13] consists of finding the parameters θ for the policy π that maximises the entropy of the distribution on the trajectories subject to (7). In case of deterministic MDPs, this effectively leads to solving the maximum likelihood problem for the demonstrated actions for a distribution which satisfies the following:

$$\mathbf{Pr}(\tau|\theta, T) = \frac{1}{Z(\theta, T)} \prod_{s_t, a_t, s_{t+1} \in \tau} \mathbf{Pr}\left(s_{t+1}|s_t, a_t\right) \exp\left(\sum_{i=1}^{k} \theta_i f_i^T\right) \prod_{i=1}^{H} T(s_t, a_t, s_{t+1})$$

where $\tau = \sum_{i=1}^{H} s_i a_i$ and $Z(\theta, T)$ is a normalisation factor The reward function, here, observes a Boltzmann distribution

To calculate the expert's expected feature counts, we use the following algorithm, which will then be formalised:

- Start from a terminal State
- Compute the partition function at each state and action to obtain the local action probabilities
- Compute the state frequencies at each time step
- Sum over the agent's state frequency for all time steps

We now formalise this algorithm

Algorithm 1: Expected Edge frequency calculation

Result: Expected Edge frequency calculation

Backward Pass

Set $Z_{s_i,0} = 0$

Recursively Compute for N iterations:

$Z_{a_i,j} = \sum_k \mathbf{Pr}\left(s_k|s_i, a_{i,j}\right) e^{(\text{reward}(s_i|\theta))} Z_{s_k}$

$Z_{s_i,0} = \sum_{a_{i,j}} Z_{a_{i,j}}$

Local Action Probability Computation

$\mathbf{Pr}\left(a_{i,j}|s_i\right) = \frac{Z_{a_{i,j}}}{Z_{s_i}}$

Forward Pass

Set $D_{s_i,t} = \mathbf{Pr}\left(s_i = s_inital\right)$

Recursively Compute for $t = 1toN$:

$D_{s_i,t+1} = \sum_{a_{i,j}} \sum_k D_{s_k,t} \mathbf{Pr}\left(a_{i,j}|s_i\right) \mathbf{Pr}\left(s_k|a_{i,j}, s_i\right)$

Summing Frequencies

$D_{s_i} = \sum_t D_{s_i,t}$

$\nabla L\left(\theta\right) = \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f_{s_i}}$

It must be noted that this procedure is very similar to value iteration

Maximum Likelihood Estimation

For Maximum Likelihood Estimation use the likelihood of observing the demonstrated data for θ as the cost function to optimise. Intuitively, this can be understood as the difference in the agent's empirical feature counts and the expert's expected feature counts. We need to find the following:

$$\theta^* = \arg\max_{\theta} L\left(\theta\right)$$

Where $L\left(\theta\right)$ is given by $\sum_{examples} \log \mathbf{Pr}\left(\tilde{\tau}|\theta, T\right)$

The gradient of $L\left(\theta\right)$ is given by

$$\nabla L\left(\theta\right) = \tilde{\mathbf{f}} - \sum_{\tau} \mathbf{Pr}\left(\tau|\theta, T\right) = \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f_{s_i}}$$

It is noteworthy that this function is convex for deterministic MDPs

3.2.2 Relative Entropy Methods

A different approach to entropy optimization in IRL involves minimising the relative entropy between two distributions P and Q over the trajectories, that is, minimising :

$$\min_{P \in \nabla} \sum_{\tau \in (SxA)^l} P(\tau) \log \frac{P(\tau)}{Q(\tau)} \tag{9}$$

One popular method for implementing this is REIRL [3]. Again, as in Maximum Entropy Methods, the distribution P is constrained to have the empirical feature counts match the expected feature counts from the trajectories, while distribution Q is obtained by sampling trajectories from a baseline policy. The baseline policy here serves as a way to provide domain specific information to the method. Knowing the transition dynamics would allow for an analytical solution, however, we can use stochastic gradient descent to solve for an approximate solution even without them

To solve (9), we can write down the dual to the problem which leads to

$$g(\theta) = \sum_{i=1}^{k} \theta_i \hat{f}_i - \log Z(\theta) + \sum_{i=1}^{k} |\theta_i| \epsilon_i \tag{10}$$

The dual problem consists of maximizing $g(\theta)$, and it is noteworthy that g in (10) is concave and differentiable everywhere except when $\theta_i = 0$. Hence, subgradient ascent can be used to maximise g. The gradients when g is differentiable are obtained as follows

$$\frac{\partial g(\theta)}{\partial \theta_i} = \hat{f}_i - \sum_{\tau \in \mathcal{T}} \mathbf{Pr}(\tau|\theta) f_i^T - \alpha_i \epsilon_i$$

where $\alpha_i = 1$ if $\theta_i \geq 0$, and $\alpha_i = -1$ otherwise Again, if the transition probability matrix T is known then the subgradient can be calculated analytically. However, if T is not known, then the subgradient can be estimated empirically using a method based on sampling trajectories based on an arbitrary policy π and then using Importance Sampling to approximate the gradient

3.3 Bayesian Methods

An important class of IRL algorithms is based on Bayesian methods, where we model a probability distribution over the set of candidate reward functions. The motivation for a probabilistic treatment of the IRL problem stems from the uncertainty arising from the underspecified nature of the problem. There may be multiple reward functions which satisfactorily explain the expert policy behaviour. This uncertainty is captured by considering a probability distribution over the set of all reward functions. Another advantage is the ability to infuse prior (domain) knowledge by imposing a prior distribution over the set of candidate functions. Thus, the mechanism of Bayesian IRL methods can be viewed upon as an update of the prior using the state-action pairs of a trajectory as *evidence*.

We make a few simplifying assumptions here:

1. The expert agent is attempting to maximize the total accumulated reward (as opposed to exploring the environment)

2. The expert agent's policy is stationary, i.e. it does not change with time or with its previous states and actions.

Let $\mathbf{Pr}(\hat{R})$ denote the prior distribution over the reward functions and $\mathbf{Pr}(\tau|\hat{R})$ denote the likelihood of the trajectory τ under the reward hypothesis $\hat{R}$. We estimate the posterior $\mathbf{Pr}(\hat{R}|\tau)$, which would quantify the likelihood of the candidate reward function $\hat{R}$ given that we have observed the trajectory τ. To this end, using the Bayes' Rule, we get that

$$\mathbf{Pr}(\hat{R}|\tau) \propto \mathbf{Pr}(\tau|\hat{R})\,\mathbf{Pr}(\hat{R}) \tag{11}$$

Under the second assumption listed above, we see that the state-action pairs observed in a trajectory are independent with respect to each other given the reward function $\hat{R}$; i.e.,

$$\mathbf{Pr}(\tau|\hat{R}) = \prod_{(s,a)\in\tau} \mathbf{Pr}((s,a)|\hat{R})$$

3.3.1 Techniques

BIRL (Ramachandran et al.) [11] In this paper, the likelihood function $\mathbf{Pr}((s,a)|\hat{R})$ is modelled using a Boltzmann distribution, with the optimal Q-value for the reward function $\hat{R}$ being used as the energy function.

$$\mathbf{Pr}((s,a)|\hat{R}) = \frac{1}{Z} e^{\alpha Q^*(s,a,\hat{R})} \tag{12}$$

where α is a scalar parameter representing the degree of confidence in the agent's optimality and Z is a suitable normalization constant.

The choice of prior depends on the nature of the problem. If we choose to be agnostic of the prior, a uniform distribution over the space of reward functions can be assumed, i.e. $-R_{\max} \leq R(s) \leq R_{\max}$ for each $s \in S$. If sparseness is preferred, a Gaussian or Laplacian prior would be used. In cases where it is known that some states have high rewards but others have a low (or negative) reward, a Beta distribution can be used as a prior with modes at the high and low ends of the reward space, i.e. $\mathbf{Pr}(\hat{R}(s) = r) = \frac{1}{\sqrt{\left(\frac{r}{R_{\max}}\right)\left(1-\frac{r}{R_{\max}}\right)}} \forall s \in S$

Active Learning (Lopes et al.) [6] This work extends the usual IRL setting by assuming that the expert agent can be queried for the optimal action at particular states. To minimize the number of "queried" actions, the optimal action for only those states is queried for which the induced posterior has maximum entropy.

Define $\mathcal{R}_{sa}(p)$ as the set of reward functions R such that $\pi_R(s,a) = p$. For each pair $(s,a) \in S \times A$, the distribution $\mathbf{Pr}(R|\tau)$ induces a distribution over p for $\pi(s,a)$. We can write:

$$\mu_{sa}(p) = \mathbf{Pr}[\pi(s,a) = p|\tau]$$

The idea is to query actions for those states for which μ_{sa} has a higher "spread", i.e. the states which have maximum uncertainty. This uncertainty is captured using the average

entropy over the actions. We choose the state s^* for which the entropy is highest, and query the corresponding optimum action from the expert agent. These newly sampled queries are thought to be more informative and helpful in learning an improved posterior.

$$H(s) = \frac{1}{|A|} \sum_a H(\mu_{sa})$$
$$s^* = \arg\max_{s \in S} H(s)$$

Gaussian Processes (Levine et al.) [5] In this paper, the reward function is set to be a nonlinear function of predetermined features ϕ, i.e. $\hat{R} = f(r, \phi)$, where r is a parameter. The function f is modelled as a Gaussian process, with the structure described by a kernel function (with parameter θ). With this description, the posterior $\mathbf{Pr}(\tau|\hat{R})$ in Eq. 11 becomes $\mathbf{Pr}(r, \theta|\tau, \phi)$. Note that

$$\mathbf{Pr}(r, \theta|\tau, \phi) \propto \mathbf{Pr}(\tau, r, \theta|\phi) = \left[\int_{\hat{R}} \underbrace{\mathbf{Pr}(\tau|\hat{R})}_{(1)} \underbrace{\mathbf{Pr}(\hat{R}|r, \theta, \phi)}_{(2)} d\hat{R} \right] \mathbf{Pr}(r, \theta|\phi)$$

Term (1) is calculated as in Eq. 12, while term (2) is a Gaussian distribution with analytically calculated mean and covariance in terms of the kernel function.

Maximum Likelihood estimation (Babes-Vroman et al.) [2] In this paper, instead of posterior estimation, the likelihood $\mathbf{Pr}(\tau|\hat{R})$ of the trajectory is directly maximized on the input data. The reward function is hypothesized to be $\hat{R}_\theta(s, a) = \theta^T \phi(s, a)$ and the likelihood of the observed trajectories is maximized with respect to θ. There is one catch: the discounted cost has a "maximization" operation which is non-differentiable. To circumvent this, a "Boltzmann exploration" operation is used to calculate the Q-values.

$$Q_\theta(s, a) = \theta^T \phi(s, a) + \gamma \sum_{s'} T(s, a, s') \bigotimes_{a'} Q_\theta(s', a')$$

where $\bigotimes_a Q_\theta(s, a) = \sum_a Q(s, a) e^{\beta Q(s,a)} / \sum_{a'} e^{\beta Q(s,a')}$ and β is a hyperparameter. The policy π_θ is, therefore, given by

$$\pi_\theta(s, a) = e^{\beta Q_\theta(s,a)} / \sum_{a'} e^{\beta Q_\theta(s,a')}$$

The log likelihood is then given by

$$\mathcal{L}(D_E|\theta) = \sum_{i=1}^{m} \sum_{(s,a) \in \tau_i} w_i \log \pi_\theta(s, a)$$

where w_i are trajectory-specific weights encoding the frequency of trajectory i. The optimal parameter is given by $\theta^* = \arg\max_\theta \mathcal{L}(D_E|\theta)$

3.4 Based on Regression and Classification

Inverse Reinforcement Learning problems can be casted into standard Machine Learning algorithms like supervised learning - regression and classification.

3.4.1 Using action-value scores

Intuitively, under an expert policy, the state-action pairs can be thought of as as a data-label pair for a classical machine learning problem. Since standard IRL problems mostly involve a large action space, this would result in a multi-class classification. The classifier learns to predict actions/labels at each state to minimize the error between actions from the expert trajectory and those from the predicted trajectory. Mathematically, the value function of this formulation can be stated as:

$$Q^{\pi}(s,a) = w^{T}\mu^{\phi}(\pi)(s,a) \tag{13}$$

This value function uses linear weighting of features vector which are the same as the feature vectors used in reward function formulation. Note that the above equation of scoring function assumes the knowledge about transition probabilities. However, in situations where transitions probabilities are unknown, a standard regression model can be fit to estimate the transiting model and henceforth learn the reward function.

3.4.2 Using Regression Trees

In situations with a large state space, the previous formulation of a global scoring function could blow up. An alternative is using a regression tree wherein a path indicates a state action space and the entire tree acts as a partition over this space. in this formulation, the following objective function is minimized

$$\min ||R^{E} - Proj_{\phi^{(i-1)}}(R^{E})|^{2} \tag{14}$$

In the above equation, $Proj_{\phi^{(i-1)}}$ refers to the projection of reward function onto the subspace spanned by features $\phi^{(i-1)}$ of the previous iteration.

4 Applications

Inverse Reinforcement Learning techniques find wide applicability in scenarios where expert behaviour is sought to be replicated or learnt, but an explicit description of the reward function is difficult to formulate. The reward function is the most succinct and transferable description of a problem objective, especially over learnt policies, which might be susceptible to errors arising due to slight variations in the environment (e.g. noise in the transition functions). However, reward functions are invariant of such changes and thus could still be useful in learning the optimal behaviour. In this section, we discuss a few possible applications of IRL techniques (some of which might be unconventional).

Self-Driving Cars Self-driving or autonomous cars, once thought to be elements in the realm of science-fiction, are now a reality. The reward function that describes optimum behaviour is difficult to design due to the large number of factors that need to be taken into consideration, e.g. how to control speed and braking around other vehicles, how to smoothly take turns or stop, how to react to a sudden pedestrian on the road, and so on. It would be much simpler to supply the agent with expert behaviour in the form of sensor measurements from a car being driven by a human as "model behaviour" which the agent is supposed to mimic. In such problems, IRL techniques would provide a much more natural solution [12].

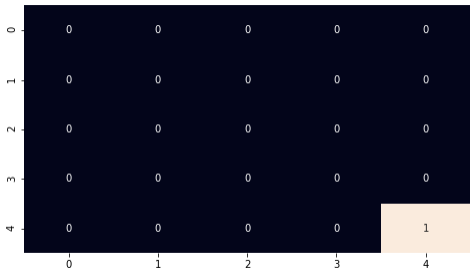
Neural Architecture Search Improving the performance of convolutional neural networks by designing better architectures is a difficult problem due to the large search space of such algorithms. Neural Architecture Search refers to the problem of automatically finding a suitable architecture. While many NAS techniques achieve sizable performance improvements, they do so at the cost of increasing complexity and inference costs. On the other hand, it is observed that human-designed networks tend to have a simpler topology and thus, greater efficiency. Therefore, IRL has been used to find architectures that are topologically similar to human-designed networks by means of a mirror-stimuli function while still obtaining improved performance [4]. The algorithm aims to find a reward function that explains the topology of the human-designed network and imitates the same, while also maintaining high target accuracy.

Detecting Troll Behaviour in Elections Digital disinformation is a major issue plaguing social media platforms, with far-reaching consequences; in particular, it is feared that malicious agents or "trolls" may skew election results. Detecting troll accounts is a challenging problem that could potentially mitigate the damage posed by such agents. In [7], Luceri et al. aim to tackle the problem by using IRL methods to understand the incentives which drive trolls' activity and use this information to distinguish trolls from among users. By modelling Twitter as an MDP, they estimated the rewards for different actions on Twitter (i.e. tweeting, re-tweeting, comments, etc.) and verified that troll and non-troll accounts have different reward profiles. While users' actions depend more heavily on interactions, the trolls are more concerned with sharing a new original tweet.

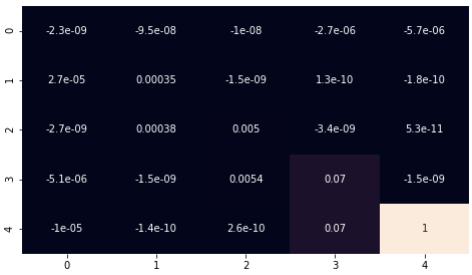
5 Experiments

We ran two of the IRL algorithms described above on a simplistic Gridworld environment. In this experiment, we assume that there are n^2 states, arranged in the form of a $n \times n$ grid. Each cell has an associated reward with landing on the cell. In addition, we assume that there is a random "wind" factor, which perturbs the next state stochastically, i.e. with a probability p, the next state is encountered randomly from among the neighbouring cells of the current state, irrespective of the action selected.

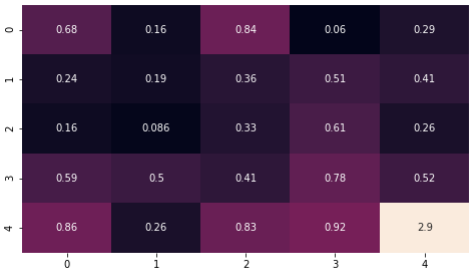
We test the two algorithms on different 'ground-truth' reward configurations and examine whether the IRL algorithms are able to infer the correct reward structure. We test the linear IRL model proposed in [10] against the Maximum Entropy model proposed in [13]. For the former, we assume that we have access to the optimal policy. We ensure this by running a



(a) True Reward Distribution

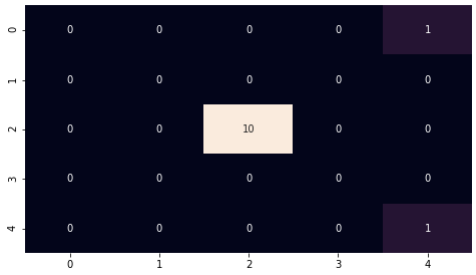


(b) Rewards from Linear IRL

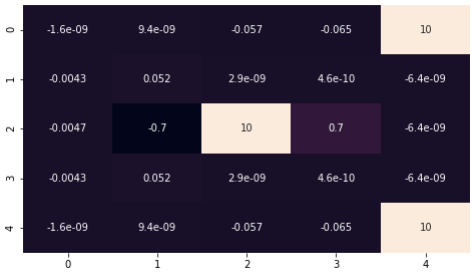


(c) Rewards from Max. Entropy

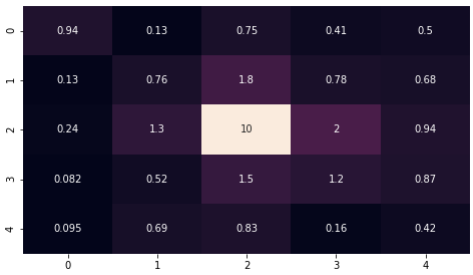
Figure 3: Heatmaps of True Reward distribution and obtained reward distributions for Setting 1



(a) True Reward Distribution

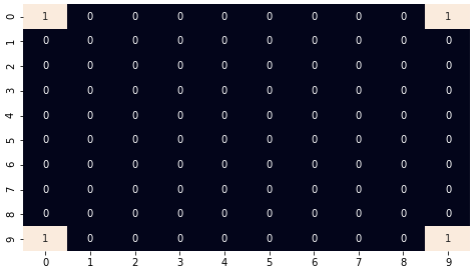


(b) Rewards from Linear IRL

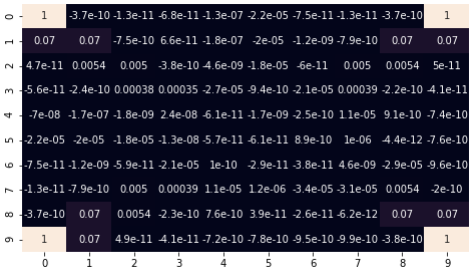


(c) Rewards from Max. Entropy

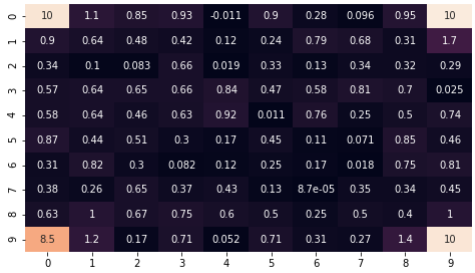
Figure 4: Heatmaps of True Reward distribution and obtained reward distributions for Setting 2



(a) True Reward Distribution



(b) Rewards from Linear IRL



(c) Rewards from Max. Entropy

Figure 5: Heatmaps of True Reward distribution and obtained reward distributions for Setting 3

value iteration over the true rewards and pass the obtained optimum policy as input to the Linear IRL model. For the latter, we assume that we only have access to the trajectories of state-action pairs encountered in the optimum policy.

We used the implementation of these algorithms from the following repository [1] and modified it to work with arbitrary GridWorld settings. We test the two models on three different settings and report results here.

Setting 1 The results from Setting 1 are shown in Figure 3. This is a simple setting where both the Linear IRL and Max. Entropy methods seem to capture the reward structure. Note that the actual scale itself does not matter, i.e. even though the state (5, 5) has true reward value of 1 and estimated value of 2.9, this does not imply that Max. Entropy method has learnt an incorrect structure, since IRL methods are invariant to scale. The Max. Entropy reward is noisier than the Linear IRL rewards, but this is to be expected since it learns the reward function from the trajectories rather than the policy.

Setting 2 The results from Setting 2 are shown in Figure 4. It is noticed that the Linear IRL method fails to distinguish the rewards at (5, 1) and (5, 5) from the reward at (3, 3). However, the Max. Entropy method fails to recognise the small rewards at (1, 5) and (5, 5).

[1] https://github.com/MatthewJA/Inverse-Reinforcement-Learning

Setting 3 The results from Setting 3 are shown in Figure 5. This is a slighlty larger state space, with the rewards at the corners of the grid. Again, it is observed that both the methods work well. Once again, we note that the Max. Entropy method achieves comparable performance even though it does not use the optimal policy itself, merely the trajectories.

6 Conclusion

In this paper, we have reviewed algorithms to implement Inverse Reinforcement Learning. We covered four main types of algorithms, namely:

- Margin optimization methods - where we find the optimal reward function by choosing a certain margin - which serves as a means to measure the efficacy of the reward function
- Entropy based methods - where we use the entropy as a measure of accuracy of the reward function
- Bayesian methods - where we model a parameterized probability distribution over the set of candidate reward functions
- Classification and regression based methods - where we cast the IRL problem into a problem which is solved by standard machine learning algorithms

We also looked some applications of IRL, particularly in self driving cars, neural architecture search and detecting of malicious agents or 'trolls' in a social setting, particularly those with far reaching implications, such as elections.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. 2004.

[2] M. Babeş-Vroman, V. Marivate, K. Subramanian, and M. Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 897–904, Madison, WI, USA, 2011. Omnipress.

[3] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. 2011.

[4] M. Guo, Z. Zhong, W. Wu, D. Lin, and J. Yan. Irlas: Inverse reinforcement learning for architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9021–9029, 2019.

[5] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in neural information processing systems*, pages 19–27, 2011.

[6] M. Lopes, F. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

[7] L. Luceri, S. Giordano, and E. Ferrara. Detecting troll behavior via inverse reinforcement learning: A case study of russian trolls in the 2016 us election. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 14, pages 417–427, 2020.

[8] D. S. Nathan D. Ratliff and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. pages 25–53, 2009.

[9] J. A. B. Nathan D. Ratliff and M. A. Zinkevich. Maximum margin planning. pages 729–736, 2006.

[10] A. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. pages 663–670, 2000.

[11] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[12] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers. Learning to drive using inverse reinforcement learning and deep q-networks. *arXiv preprint arXiv:1612.03653*, 2016.

[13] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.