Inverse Reinforcement Learning – Max Margin, Max Entropy, and Bayesian IRL

April 1, 2021 by Batuhan Altundas, Yuuna Hoshi, Reagan Kan and Marcus McGuire

Introduction

In the field of Artificial Intelligence and specifically in Robotics, task decisions are made based on some Utility Function, which provides a reward for each action that a robot can choose from. But what if the instructor does not know about the rewards? What if the user does not know about how to provide an effective Reward Function? Or, what if the user is working on a field that is not related to Robotics but they want to use robots to help their work? The solution for the lack of a Reward Function is to create a system that can draw out its own rewards for each action based on a few demonstrations of what the robot is meant to do, provided by the user. Therefore the main challenge is to have a model that can use Demonstrations provided by users with different backgrounds, extract the Reward Functions automatically and achieve high performance without relying on the niche set of skills that are required to program the robots. Inverse Reinforcement Learning aims to allow greater accessibility of Robots to humans as a whole, instead of the limited environment of Robotics Labs.

Inverse Reinforcement Learning (IRL) automates the learning of the rewards used in Reinforcement Learning. In reinforcement learning, the reward function is defined along with the state and actions the agent can take, and the agent learns to maximize this reward function. However, modeling this reward function can be difficult in many situations as the rules are not so defined. For example, even though many people know how to drive a car, explicitly formulating this as a reward function to maximize is not feasible. There are many factors to consider such as traffic rules, what to do when pedestrians are approaching, and when and how to change lanes. Inverse reinforcement learning tries to derive an unknown reward function from observing expert behavior. The assumption for IRL is that expert demonstrators are trying to maximize some underlying reward function and it is possible for the robot to learn from provided demonstrations, extracting the rewards automatically instead of requiring some unknown reward function, making Inverse Reinforcement Learning more versatile than Reinforcement Learning.

This Blog is intended as an introduction to Inverse Reinforcement Learning, its origins and uses by presenting three primary forms of IRL, Maximum Margin IRL (Abbel et. al. 2004), Maximum Entropy IRL (Ziebart et. al. 2008) and Bayesian IRL (Ramachandran et. al. 2009). The contents are divided into the following sections:

1. Related Works and Topics

Inverse Reinforcement Learning builds on Reinforcement Learning and Markov Decision processes. The main purpose of this method is to automate the Reward Function generation. The problem of inverse reinforcement learning (IRL) was first defined as the following problem:

- Given the measurements of an agent's behavior, its sensory inputs, and the environment model, determine the reward function that the agent is trying to optimize (Russel, 1998).

This work later led to the creation of three primary forms of IRL, Maximum Margin IRL (Abbel et. al. 2004), Maximum Entropy IRL (Ziebart et. al. 2008) and Bayesian IRL (Ramachandran et. al. 2009).

2. Method

A Brief Introduction to MDP (Gombolay and Tambwekar, 2021)

Inverse Reinforcement Learning is derived from Reinforcement Learning, which is described in a Markov Decision Process (MDP) Environment. The MDP Environment is made up of 5 concepts:

1. **States**: A *state*, s, is any predefined momentary instance in the world that an agent can exist in. For the rest of this post, we will use the variable S to represent the set of all possible states in the world with $s \in S$ referring to an individual state.
2. **Actions**: An *action*, a, is an event facilitated by the agent that can transition you from one state to another provided that such a transition is possible MDP. We will use to represent the set of all possible actions in the world, with $a \in A$ referring to an individual action. We note that actions may not have deterministic consequences. For example, flipping a coin may not give you the same result each time! The degree to which actions have deterministic effects is described by the *transition function*.
3. **Transition function**: The transition function is a function that defines the probability of moving to a specific next state, given your current state and a valid action. The transition function, T, is mathematically defined as follows, $S \times A \times S' \rightarrow [0,\ 1]$.
4. **Reward**: The *reward function* specifies a real number value that defines the efficacy or a measure of "goodness" for being in a state, taking an action and landing in the next state. Similarly to the transition function, the reward is defined as follows, $R\colon S \times A \times S' \rightarrow \mathbb{R}$. Note that the state you end up in may be uncontrollable since state transitions can be dynamic.

5. **Discount Factor**: The discount factor can be specified using γ, where $\gamma \in [0, 1)$. Note the non-inclusive upper bound for the discount factor (i.e., $\gamma \neq 1$). Disallowing $\gamma = 1$ allows for an MDP to be more mathematically robust. Specifically, the goal for RL algorithms is often to maximize discounted reward across time. Consider the case of an infinite horizon MDP (i.e., the MDP never ends) in which the rewards are always positive. If the discount factor, γ, is equal to 1, then the sum of future discounted rewards will be infinite, making it difficult for RL algorithms to converge (i.e., know when they can stop determining which actions should be taken in each state).

Basic Principle behind Inverse Reinforcement Learning

IRL is designed to find some reward function with respect to an 'optimal' demonstration provided. IRL tries to derive the Optimal Policy based on the MDP equation, with the primary idea of the optimal demonstration having higher rewards than any other demonstrated policies, represented with the optimization equation:
Find R^* s.t.

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi^*\right] \geq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi\right], \ \forall\pi$$

Derived from the Value function of Value Iteration, Inverse Reinforcement Learning tries to optimize R^*, which is the Demonstrator's Reward Function, using the discount factor γ^t, the Sequence of States defined at time t $\{s_t, \forall t\}$, the demonstrator's policy π^*, and the set of learner's policies π.

The Basic Principle behind Inverse Reinforcement Learning is to find a policy that has the reward for human policy, greater than or equal to any policy that the robot can come up with. This however leads to several problems, such as:

- R = 0 is a solution to the optimization problem above, leading to the Reward Ambiguity Problem since reward should not be equal to 0.
- Trainer has access to only the demonstrations that make up the "traces" or "trajectories" defined as $D = \{\tau\}$ where trajectories are a set of state action pairs defined as $\tau = \{< s_t, a_t >, \forall t \in \{1, ..., T\}\}$, instead of having τ being drawn from π^*
- We do not have direct access to the optimal policy, π^* or a way to ensure that given demonstration is not suboptimal.
- Assumes that we can easily enumerate all possible π

Linear Cost Inverse Reinforcement Learning: Feature Based Reward Function

The most basic form of IRL assumes a linear cost for the features. In Linear Cost IRL, the reward function can be computed with respect to policy at state s $\varphi(s)$ and weights associated with that state policy, simplifying the Reward optimization through:

$$R(s) = \vec{\varpi}^T \vec{\phi}(s) \;\; ; \phi : s \to R^d ; \omega \in R^d$$

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right] := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \vec{\varpi}^T \phi(s_t) \mid \pi\right] = \vec{\varpi}^T \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right] = \vec{\varpi}^T \mu(\pi)$$

where μ is the expected discounted feature count following policy $\pi(\mu : \pi \to R^d)$. μ is also referred to as the feature expectations.
Therefore, IRL can be defined as:

$$\text{Find } \omega^* \text{ s.t. } (\omega^*)^T \mu(\pi^*) \geq (\omega^*)^T \mu(\pi), \; \forall \pi \in \Pi$$

Advantages:

- $\mu(\pi^*)$ can be easily estimated empirically with Monte Carlo samples of expert trajectories.
  - $\mu(\pi) \approx \frac{1}{n} \sum_{t=0}^{\infty} \gamma^t \varphi(s_t)$ where $a_t \sim \pi(.|s_t)$, $s_{t+1} \sim \pi(.|s_t, a_t)$, $s_0 \sim p(.)$
- The number of expert demonstrations, m, needed for a good estimate is mathematically proven to be proportional to $|\phi|$ (Abbeel et. al. 2004).
  - Since the number of demonstrations, m, is proportional to $|\phi|$, m is NOT proportional to the complexity of the expert policy nor the size of the state space.

Disadvantages:

- The reward representation is restricted to be linear over features. Therefore, expected discounted sum of feature values or feature expectations are dependent on state visitation distributions, making the policy skewed towards states that are most common in demonstrations (Fragkiadaki, K. 2017).

Max Margin IRL

The idea behind Max Margin IRL is to create a maximum-margin hyperplane between policies in feature space, therefore classifying policies based on their optimality (Fragkiadaki, 2017). Therefore, the IRL equation can be represented with the equation:

$$\text{Find } \omega^* = argmin_{\omega} ||\omega||_2^2$$
$$s.t. (\omega^*)^T \mu(\pi^*) \geq (\omega^*)^T \mu(\pi) + 1 \; \forall \pi$$

The given equation is similar to Support Vector Machines (SVM), therefore it is possible to intuitively solve the IRL problem through SVM. Given a set of linearly separable data points with binary labels (Label +1 or Label -1), a Support Vector Machine finds the hyperplane that bisects the points such that the points on either side of the hyperplane have the same label. For example, in the figure below, the blue and orange points should be given labels +1 and -1 respectively. A SVM would try to find a hyperplane, or a line in this 2D case, between the blue and orange points.

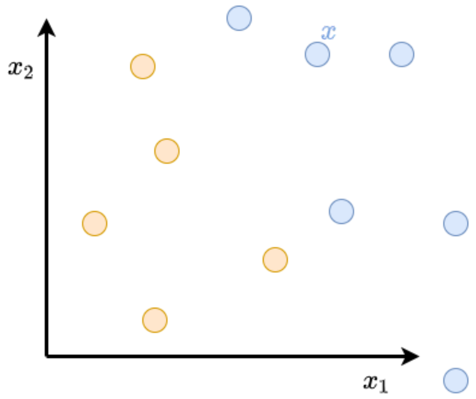

Fig. 1: Example data points in 2D.

A hyperplane can be defined as the set of points that satisfy $\omega^T x + b = 0$, where ω is the normal vector of the hyperplane. Then, two spaces created by the hyperplane are characterized by the set of points $\omega^T x + b > 0$ and $\omega^T x + b < 0$. With this definition, determining which side of the hyperplane an arbitrary point x falls on is straightforward; compute $sign(\omega^T x + b)$. For example, consider the blue point x in the figure below. The red line is the SVM hyperplane. To classify point x, the SVM projects the point x onto the vector ω; that is the purple point $\omega^T x$. The $sign(\omega^T x + b)$ evaluates to +1 because the purple projection is on the right-hand side of the hyperplane. The remaining blue and orange points are labeled similarly until all blue points are labeled +1 and all orange points are labeled -1.

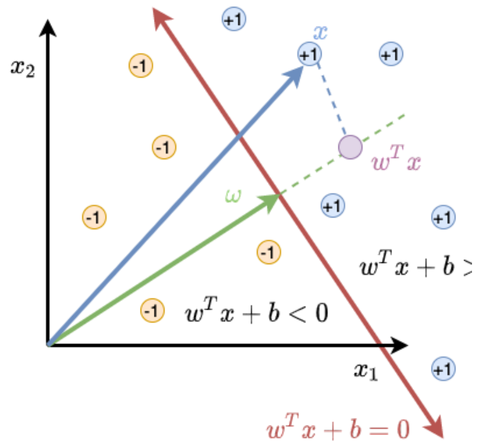

Fig. 2: A 2D example of SVM classification.

There can be many valid separating hyperplanes. SVMs define the “best” hyperplane as the one that maximizes the *margin* between the hyperplane and the closest data points. More formally, the SVM solves the following optimization problem, which is equivalent to maximizing the *margin* (Ma & Ng 2019):

$$\omega^* = argmin_{\omega} \|\omega\|_2^2 \ \ s.t. \ y(\omega^T x + b) \geq 1, \ \forall (x, y)$$

The SVM problem can be used to solve the Max Margin IRL problem, where the data points are the policy feature expectations. Label +1 is given to the expert policy feature expectations and Label -1 is given to all other policies. Under this configuration, the SVM constraints give the following inequality for the expert policy:

$$1 \leq 1(\omega^T \mu(\pi^*) + b) \text{ (expert constraint)}$$

And, the SVM constraints give the following inequality for non-expert policies, which can be reworked to match the Max Margin constraint when coupled with expert constraint:

$$-1\,(\omega^T \mu(\pi) + b) \geq 1 \ \forall\pi \text{ (non-expert constraint)}$$
$$\omega^T \mu(\pi) + b + 1 \leq 1 \ \forall\pi \text{ (algebra)}$$
$$\omega^T \mu(\pi) + b + 1 \leq \omega^T \mu(\pi^*) + b \ \forall\pi \text{ (incorporate expert constraint)}$$
$$\omega^T \mu(\pi^*) \geq \omega^T \mu(\pi) + 1 \ \forall\pi \text{ (algebra; now matches Max Margin inequality)}$$

Now, the SVM problem matches the Max Margin problem, i.e. SVMs can solve the parameters for the feature based reward function.

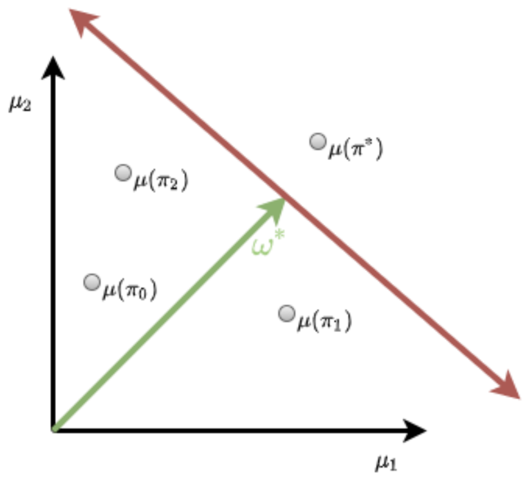

Fig. 3: Separation between expert feature expectations and non-expert feature expectations

Algorithm 1: Constraint Generation (Abbel et. al. 2004)

```
1. Estimate $\mu(\pi^*)$ using $k$ expert demonstrations, where $k \sim |\phi|$
2. Initialize $\pi_0$ and add to empty policy set $\Pi$
3. Solve for $\omega^*$ using an SVM. Optional: add slack variables / distance function
4. Solve for a new policy $\hat{\pi}$ using $R(s) = (\omega^*)^T \phi(s)$. Add $\hat{\pi}$ to $\Pi$
5. If $(\omega^*)^T \mu(\pi^*) \geq (\omega^*)^T \mu(\pi) + 1 \; \forall \pi \; in \; \Pi$ return $\omega^*$
6. Go to Step 3
```

Algorithm 1 puts together the concepts presented so far to solve the overarching IRL problem, which is to find a reward function that motivates an expert demonstrator. Under the assumption that the reward is a linear weighting of state features, the Max Margin formulation of the IRL problem becomes:

$$\text{Find } \omega^* \text{ s.t. } (\omega^*)^T \mu(\pi^*) \geq (\omega^*)^T \mu(\pi) + 1 \; \forall \pi$$

In other words, the reward parameters should result in the expert reward being greater than all non-expert rewards. Step 1 of the algorithm solves for $\mu(\pi^*)$ which sets up the left-hand side of the Max Margin inequality. Steps 2 - 4 set up the right-hand side. Specifically, Step 1 initializes a collection of non-expert policies. Step 3 leverages the SVM analogy to solve for linear reward function parameters. Given those parameters, Step 4 leverages a Reinforcement Algorithm to solve for a new policy $\hat{\pi}$ to be added to a collection of non-expert policies. Note, any reinforcement algorithm can be used in Step 4. Finally, Steps 5-6 check that the Max Margin constraint is met and return the reward parameters. If the constraint is unmet, the algorithm tries again after adding another non-expert policy to its collection.

The optional note in Step 3 refers to slack variables, which are a way to handle suboptimality, and a distance function, which is key to the "Standard Prediction" Max Margin formulation. These ideas will be presented in the subsequent sections.

Suboptimality

Suboptimality can come from suboptimal expert demonstrations and/or suboptimal feature space. In the SVM analogy, this means the expert policy feature expectations are not linearly separable from all other policy feature expectations. The figure below shows an example of this in a 2D space. In this situation, it is impossible to have a 2D hyperplane that can linearly separate the expert policy (π^*) from the other sub-optimal policies (π_0, π_1, π_2).

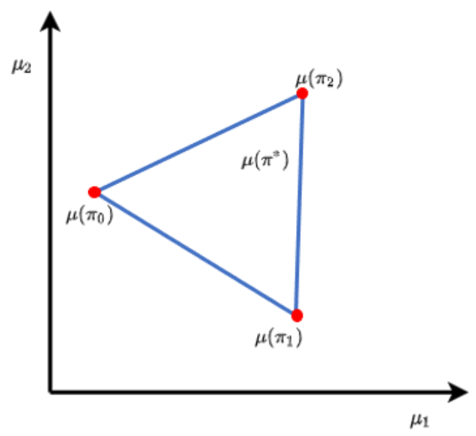

Fig. 4: Expert policy cannot be linearly separated from other policies in feature space, indicating suboptimality.

In Figure 4, the expert policy, π^* , cannot be linearly separated from other policies in the feature space. The presence of $\mu(\pi_2)$ shows that there is a policy that is more effective than the supposed expert policy, which indicates the presence of some form of suboptimality.

Solution to Suboptimality: Slack Variables

$$\text{Find } \omega^* = argmin_{\omega} ||\omega||_2^2 + C \sum_{\pi} \xi^{\pi}$$

$$s.t.\ \omega^T \mu(\pi^*) \geq \omega^T \mu(\pi) + 1 - \xi^{\pi} \ \forall\pi \text{ where C is a tuned hyperparameter.}$$

Aside from engineering better features and perfecting demonstrations, a solution to suboptimality is to add slack variables to the inequality constraint in the Max Margin problem. Each policy receives a slack variable $\xi^{\pi} \geq 0$. In the 2D example, π_2 would receive a non-zero slack variable to allow a violation of the constraint inequality. To prevent the slack variables from growing too large, the sum of the slack variables is included in the minimization, similar to a regularization term. The previous Max Margin formulation can be seen as a special case of the slack variable version with all slack variables set to 0.

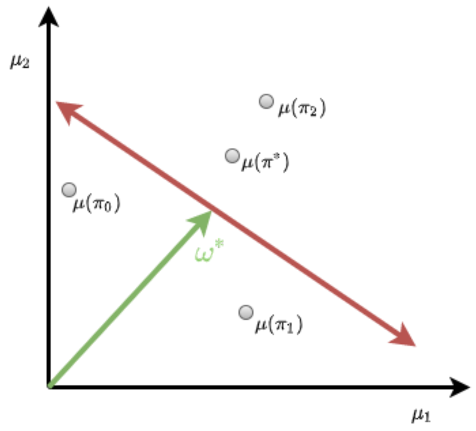

Fig. 5: the feature expectations of a non-expert policy, $\mu(\pi_2)$, is allowed to cross the decision boundary (red) because it has a non-zero slack.

"Standard Prediction" Max Margin Formulation

$$\text{Find } \omega^* = argmin_{\omega} ||\omega||_2^2 + C\sum_{\pi} \xi^{\pi}$$
$$s.t.\ \omega^T\mu(\pi^*) \geq \omega^T\mu(\pi) + 1 - \xi^{\pi} + m(\pi, \pi^*)\ \forall\pi$$

The main idea behind the Standard Prediction Max Margin is that the performance margin between the expert policy and another policy should be related to how "different" the two policies are. This notion of difference is represented by the $m(\pi, \pi^*)$ term and will move the hyperplane away from the policies that are different from the expert. An example of this function is the number of states that π and π^* disagree on, i.e. $\pi(s) \neq \pi^*(s)$.

Max Entropy IRL

In Maximum Entropy IRL, instead of the Maximum Margin, an Entropy variable is used to handle the Reward Ambiguity. Based on Information Theory, the Maximum Entropy IRL tries to maximize the entropy of the distribution over possible trajectory paths subject to the feature constraints from observed data, to maximize the likelihood of the observed data under the maximum entropy (exponential family) distribution (Jaynes 1957), which in turn allows for the selection of the trajectory with greatest Information Gain, addressing the problem of Reward Ambiguity using the following equation:

$$\omega^* = argmax_{\omega} \sum_{\tau \approx Dem} log\ P(\tau|\omega, T)$$

where, τ is the trajectory from the Demonstrator represented as a set of state action pairs in the form of $\tau = \{ < s_0, a_0 >, < s_1, a_1 >, ..., < s_n, a_n >\}$ and T is the transition function. The purpose of Max Entropy IRL is to make the trajectory from Demonstration very likely to happen.

1) Using Gradient Descent over Model Parameters

We use the Gradient Descent approach to improve the model. To achieve this, we use value iteration for every step of the gradient descent algorithm where Loss can be represented as:

$$\begin{aligned}\nabla L(\omega) &= \left[\frac{1}{m}\frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{n}\phi(s_i)\right] - \sum_{\tau} Pr(\tau|\omega, T)\\ &= \mu(\pi^E) - \sum_{s_i} D_{s_i}\phi(s_i)\end{aligned}$$

Given the Expected Edge Frequencies and the Loss Function, Gradient can be calculated with ease. In order to use the gradient descent approach, D_{s_i} is calculated by enumerating each possible path with the following pseudocode:

Algorithm 2: Expected Edge Frequency Calculation (Ziebart et. al. 2008)

```
A) Backward Pass
    1) $z_{s_i}^{(0)} \leftarrow 1, \forall i, j$
    2) For l = 1 to N
    3)     $z_{a_{i,j}}^{(l)} \leftarrow \sum_k T(s_k|s_i, a_j)\, e^{\omega\varphi(s_i)}\, z_{s_k}^{(l-1)}, \forall i, j$
    4)     $z_{s_i}^{(l)} \leftarrow \sum_{a_{i,j}} z_{a_{i,j}}^{(l-1)}, \forall i$
B) Local Action Probability Computation
    5) $P(a_{ij}|s_i) \leftarrow \frac{z_{a_{ij}}^{(N)}}{z_{s_i}^{(N)}}, \forall i, j$
C) Forward Pass
    6) $D_{s_i}^{(0)} \leftarrow P(s_i == s_{initial})$
    7) For t = 1 to N
    8)     $D_{s_t}^{(t)} \leftarrow \sum_{a_{ij}}\sum_k D_{s_k}^{(t-1)} P(a_{ij}|s_i) T(s_k|s_i, a_{ij})$
D) Summing over Frequencies
    9) $D_{s_i} \leftarrow \sum_t D_{s_i}^{(t)}$
```

In the Backwards Pass Step of Algorithm 2, we calculate the probability mass associated with each branch through a Backwards Iteration. The equation used calculates the

Entropy of each action and state based on the demonstrated trajectories. In the Local Action Probability Computation Step, we calculate the partition function for each action and state pair based on the previously computed probabilities for each state and action. The partition function yields local action probabilities which are used in the Forward Pass Step to compute state frequencies in each timestep. The State Frequencies are then combined for each branch (Ziebart et. al. 2008).

Bayesian IRL

Bayesian IRL attempts to maximize the probability of the reward function given a demonstrator rather than maximizing trajectory given reward like in other IRL methods. Bayesian IRL assumes there is a demonstrator, χ, operating in a markov decision process, who is attempting to maximize their total accumulated reward from function, R, with a stationary policy. With these assumptions, it is possible to model the probability of each state action pair from X's demonstration with the function:

$$Pr_{\chi}((s_i, a_i)|R) = \frac{1}{Z_i} e^{\alpha\chi Q^*(s_i, a_i, R)}$$

where α is a parameter that represents the coincidence that χ selects the best action from their Q^* function. Because the deomstrator's policy is assumed to be stationary, the probability of the whole demonstration, O_{χ}, can be given as:

$$Pr_{\chi}(O_{\chi}|R) = \frac{1}{Z} e^{\alpha\chi E(O_{\chi}, R)} \text{, where } E(O_{\chi}, R) = \sum_i Q^*(s_i,\ a_i,\ R)$$

This likelihood is essentially a boltzmann-type distribution with energy $E(O_{\chi},\ R)$ and temperature $\frac{1}{\alpha\chi}$. From this likelihood, we derive $Pr(R|O_{\chi})$ by applying the Bayes Rule:

$$Pr(R|O_{\chi}) = \frac{Pr_{\chi}(O_{\chi}|R)P_R(R)}{Pr(O_{\chi})} = \frac{1}{Z'} e^{\alpha\chi E(O_{\chi}, R)} P_R(R)$$

The main reason for using this model formulation is that it allows for parameterization of additional information. In particular, the alpha parameter, which controls the temperature, is determined by the skill of a demonstrator. A lower temperature demonstration will have lower overall effect and confidence in the model. This allows for suboptimal demonstrators to be included in the model's training data and still have the model achieve good performance. This model formulation also allows for parameterization of a prior of rewards which can help incorporate knowledge of the problem space into the model.

When evaluating the loss for a Bayesian IRL model on a reward learning task, either linear or squared error are commonly used:

$$L_{Linear}(R,\ \widehat{R}) = \|R - \widehat{R}\|_1$$
$$L_{SE}(R,\ \widehat{R}) = \|R - \widehat{R}\|_2$$

For apprenticeship learning task, Bayesian IRL defines the following policy loss function:

$$\mathrm{L}^{p}_{policy}(R,\ \pi) = \left\| V^{*}(R) - V^{\pi}(R) \right\|_{p}$$

Where $V^{*}(R)$ is the vector of optimal values for each state achieved by the best policy for R and p is any chosen norm. To solve a Bayesian IRL problem, the goal is to find the policy that minimizes the expected policy loss for the posterior distribution for R. A direction minimization of policy loss is a hard problem to compute, so instead the optimal policy for the mean reward function can be computed. These lead to the same policy and a detailed proof of why this is the case is available in Ramachandran, D., and Eyal A. 2009. However, in order to compute the optimal policy for the mean reward function, the mean reward function itself must be computed, which is not trivial. Bayesian IRL uses a markov chain monte carlo sampling algorithm known as PolicyWalk to generate samples from the posterior distribution and returns the mean reward function from the samples that is then treated as the mean reward function.

Algorithm 3: Policy Walk (Ramachandran et. al. 2009)

```
1. Pick a random reward vector $R \in \mathbb{R}^{|S|}/\delta$
2. $\pi := PolicyIteration(M,\ R)$
3. Repeat:
    a. Pick a reward vector $\overline{R}$ uniformly at random from the neighbours of $R \in \mathbb{R}^{|S|}/\delta$
    b. Compute $Q^{\pi}(s,a,\overline{R})$ for all $(s,a) \in S,\ A$.
    c. If $\exists(s,a) \in (S,A)$, $Q^{\pi}(s,\pi(s),\overline{R}) < Q^{\pi}(s,a,\overline{R})$
        i. $\overline{\pi} := PolicyIteration(M,\overline{R},\pi)$
        ii. Set $R := \overline{R}$ and $\pi := \pi$ with probability $min\{1,\ \frac{P(\overline{R},\overline{\pi})}{P(R,\pi)}\}$
       Else
        i. $R := \overline{R}$ with probability $min\{1,\ \frac{P(\overline{R},\pi)}{P(R,\pi)}\}$
4. Return $R$
```

PolicyWalk moves along a markov chain and keeps track of the optimal policy for the current reward vector. It starts with a random reward vector and optimal policy for that vector. Then it picks a random similar reward vector. It then computes the Q function for the MDP with that new reward vector. If there is any state-action pair where the old policy is not optimal with the new reward vector, then it may update the reward and policy based on a probability from the posterior distribution. This step of sampling a new reward vector and checking for updates is repeated until a satisfactory reward vector is found.

Compared to other methods of IRL, Bayesian IRL has several advantages. These advantages come from the additional parameters that are available in the likelihood function. The α parameter allows for information about a demonstrators skill to be included in the model. In situations where there are suboptimal demonstrators, Bayesian IRL can still perform well because it acknowledges in the model that demonstrators will not always take the correct action. The $P_R(R)$ parameter allows for a prior of rewards to be included into the model. This helps incorporate further problem-level information into the model that was not present in just demonstrations, which often leads to better performance. However, since these extra parameters are not optional, they could also lead to worse performance if they are not reasonably estimated.

3. Evaluation

IRL works well for situations where the reward function is difficult to formulate, or in scenarios where learning the reward function itself is a goal. This makes IRL extremely effective in the field of Learning from Demonstration, where an expert can train a robot to do a task. Even outside of computer science related fields, the idea of IRL can be applied to other fields such as physical sciences and control theory. In the field of physical science, inverse problem theory works with inferring model parameters from a description of a physical system. In control theory, there has been work done on recovering an objective function for deterministic linear systems.

	Advantages	Disadvantages
Linear Cost Feature Based Reward IRL	- Number of demonstrations required to estimate expert policy is linear to the number of dimensions of state feature space.	- Sensitive to Reward Ambiguity - Sensitive to suboptimality
Max Margin IRL	- Relaxed constraints allowing for suboptimality with the use of the slack variable - Can be optimized using an SVM	- Assumes linear rewards - Needs to solve for a new policy from scratch in each iteration.
Max Entropy IRL	- Model is built on the probabilistic model of demonstration paths, making it more accurate.	- Requires knowledge of environment dynamics - Requires data points to be very similar
Bayesian IRL	- No suboptimality with good knowledge of demonstrator skill and reward prior	- Assumes linear rewards - Scales poorly in large environments - Requires some amount of knowledge of demonstrator skill and reward prior

4. Conclusion

There are many challenges remaining in the field of IRL. Suboptimality of demonstrations is one of the key challenges faced when using IRL. The lack of a way to validate if the demonstration is optimal or not makes the application of IRL to critical applications impossible. Moreover, feature selection remains a challenge for engineering effective models through IRL, often requiring hand engineered features so that policies are linearly separable, which requires a lot of effort or the creation of a feature extractor model. Some other problems are that many existing methods do not have provable analysis of the time complexity of their methods, and do not have comparisons with other methods with this metric, and that there is a lack of testbed domains that can be used for evaluation of IRL methods (Arora et al, 2019).

The key advantage of IRL compared to Reinforcement Learning is the removal of the need for the demonstrator to have knowledge about the programming methodologies used as well as the removal of the need for the roboticist using the IRL to have expert knowledge about the field of application. A simple interface to allow for the training of the robot by experts. This makes extraction of the domain expertise, allowing for the deployment of robotic automation in a wide range of areas in short timeframes through automated domain expertise extraction. IRL also makes the application−specific nuances to be scalable.

5. Bibliography

Gombolay, M. (2021). Interactive Robot Learning Lecture and Lecture Notes Lectures 5 and 6

Abbeel, P., & Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of International Conference on Machine learning.*

Ziebart, B.D., Maas, A., Bagnell, J.A. and Dey, A.K., 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI).*

Ramachandran, D., and Eyal A. 2009. Bayesian inverse reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp 2586-2591).

Ho, J. and Ermon, S., 2016, December. Generative adversarial imitation learning. In Proceedings of the 30th International Conference on Neural Information Processing Systems (pp. 4572-4580).

Arora, S. and Doshi, P., 2019. A survey of inverse reinforcement learning: Challenges, methods and progress. Artificial Intelligence, Volume 297, 2021, 103500.

Ma, T. & Ng, A. 2019. CS229 Lecture Notes, Stanford University, delivered 21 April 2019. http://cs229.stanford.edu/summer2019/cs229-notes3.pdf

Learning agents for uncertain environments (extended abstract).
Russel, S., 1998. Proceedings of the Eleventh Annual Conference on Computational Learning Theory - COLT 98.

Jaynes, E. T. 1957. Information theory and statistical mechanics. Physical Review 106:620–630.

Gombolay, M. and Tambwekar, P. 2021. Bootcamp Summer 2020 Week 3 – Value Iteration and Q-learning. CORE Robotics Lab Bootcamp Blogpost.
https://core-robotics.gatech.edu/2021/01/19/bootcamp-summer-2020-week-3-value-iteration-and-q-learning/

Fragkiadaki, K. 2017. 10703 Deep Reinforcement Learning and Control Imitation Learning II Slides. CMU. https://katefvision.github.io/