2. Background and Related work

This part introduces theoretical background and basics related to this work.

**2.1. Reinforcement Learning**

The Artificial Intelligence popular image and its various current applications depend on the availability of underlying data from which patterns can be recognized; these patterns are then used to predict or act on future data. This approach seems intuitive for the application areas where data is at the core and environment in which application will act is static e.g. image recognition. But this approach seems inadequate when it comes to applications such as game playing, car driving etc. which are more dynamic in nature.

Reinforcement Learning (RL) is the technique of learning from interaction with the surrounding environment. This approach seems very natural as this is the way we Humans learn to live and progress in our environment. E.g. Mastering car driving involves starting driving with basic knowledge, receiving negative criticism for mistakes such as harsh turns or urgent braking from tutor or positive comments for perfect lane changing. [7] details the way how RL agent outperforms human in Atari game environment. This idea of reinforcement is also supported by the work in behavioural studies [2].

Below the basic literature terms are explained on which RL is based on:

*Markov Property:*

At any point in time, the future state only depends upon the current state and not on the states/paths followed to reach the current state.

*Markov Chain:*

The sequence of random states achieved by following the *Markov Property*

*Stochastic Environment:*

The environment in which taking action A from state S does not always leads to the same next state S’. Environment is said to have possess Stochasticity.

Reinforcement learning is formally defined using Markov Decision Process (MDP) as [1]:

MDP consists of tuple (S, A, T, 𝛾, R) where

S: finite set of environment states

A: finite set of agent actions possible in the environment

T: transition probability distribution i.e. a function T(st, at, st+1) giving probability of landing in state st+1 from st by taking action at.

R: reward function R(s) giving reward values for reaching state s.

𝛾: discount factor to discount the future rewards.

Some other concept which are derived from the basic RL definition and are frequently used in literature are illustrated below:

*Policy* is the function which takes current state and return action to be taken*;* defined by π: S -> A

There exists the *optimal policy* π\*, by following which the agent has higher chances of achieving the maximum reward as compared to other *suboptimal policies.* An RL agent’s objective is to determine a policy as close as to the optimal policy in terms of expected rewards.

*Value function:*

It associates a value with every state representing expected reward starting from that state and following the policy.

Vπ(s) = E[R(s1) + 𝛾 R(s2) + 𝛾2 R(s1) ….. | π]

The future rewards are discounted by the factor of 𝛾 for two reasons: 1. To avoid expected total reward to sum up to infinity 2. to make agent seek immediate reward rather than the rewards in future

*Optimal Value function* is the value function associated with the optimal policy i.e. it has highest expected reward.

V\*(s)= max Vπ(s), ∀ s ∈ S

*Q-value:*

Unlike the value function which denotes profitability of being in a particular state, Q-values represents quality associated with each possible action from a state. Optimal Q-value function is defined using what is known as Bellman equation as,

Q\*(s, a) = R (s, a) + 𝛾 (∑s’ (T(s, a, s’) V\*(s’)))

max a’ Q (s’, a’)))

Using this definition optimal value function can be restated as,

V\*(s)= max a Q\*(s, a)

V\*(s)= max a [ R (s, a) + 𝛾 (∑s’ (T (s, a, s’) V\*(s’))) ] .. (1)

*Value Iteration* iteratively solves for the equation (1) to converge to the optimal value function.

But as an RL agent only needs the policy, instead of the actual state values, to decide on the actions to take, we can stop iterating when policy converges. This approach is called *policy iteration.* From above illustration it is clear that both these approaches require environment model to compute the optimal values i.e. these methods are suitable for the model-based agent (also called offline learning). For the situation when the environment model is not available, a modified version of the above equation *viz* Temporal Difference (TD), is used to learn action qualities. The basic idea behind TD Learning is to iteratively update Q-values using the interaction with the environment.

TD (s, a) = (R (s, a) + 𝛾 maxa’ Q (s’, a’)) – Q (s, a)

Qt (s, a) = Qt-1 (s, a) + α TDt (s, a)

where α: learning rate to control how fast the values change

As can be seen from the above equation, upon each new interaction with the environment agent updates the Q-values difference it observes. This equation is bound to converge after finite iteration in non-changing environment and so we can use it to find out the policy. If environmental conditions change in the middle, it has the ability to adapt to them.

Summarizing all the above points, Reinforcement Learning can be depicted with the below image where at any time step agent senses the environment, takes an action according to its learn policy at that time and again senses the new state and the reward after taking the action.



**2.2. Reward Engineering Problem with Reinforcement Learning**

From the definition of RL it is clear that effectiveness of RL agent depends on how accurately it learns the policy to act. The closer it gets to the optimal policy the better. However, it can also be seen from the definition that the learnt policy heavily depends on the reward function, which is stated manually. A slight change in the reward function has the potential to heavily change the policy of the agent. It also decides ‘How quickly’ and ‘What things’ an agent will learn while training. As Pieter Abbeel and Andrew Ng. states that,

*“the entire field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy or the value function, is the most succinct, robust, and transferable definition of the task”* [4]

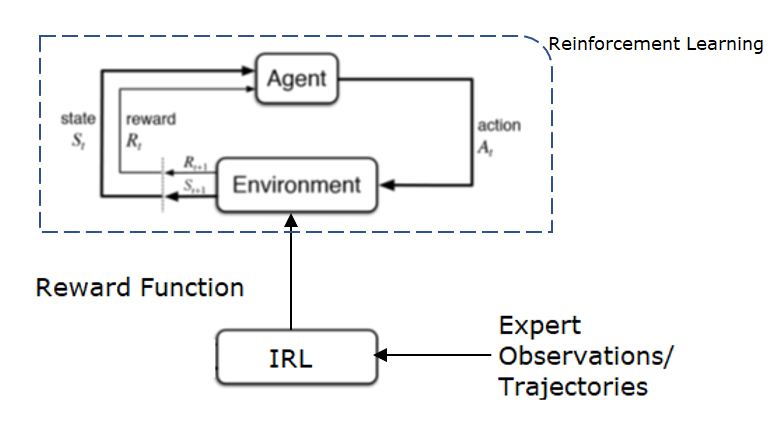
A study of the successful RL application till date e.g. [8] [9], shows that most of them are the domains where environment can readily provide the reward. E.g. Atari game where winning is rewarded positively while losing rewarded negatively. But in the real world, it is often not clear at all what the reward should be and there are rarely intrinsic reward signals such as game score. To continue with the car driving example illustrated in the above section, consider how many possible parameters are there to be a good driver e.g. lane changing, safe overtaking, understanding traffic signals and other vehicles’ signal, safe turning, maintaining safe distance and many more. To add to the complexity there might also be possible relationships between them e.g. safe distance for different speed. Specifying the perfect reward function means correctly assigning weights to all these parameters. Most of the time these weights are manually altered until performance is improved as required. Clearly, considering the amount of computational power and time required to train the RL agents, such trial-error approach is only suitable where reward function is small enough like Atari games and not where it is high dimensional.

**2.3. Inverse Reinforcement Learning**

Inverse Reinforcement Learning (IRL) is a technique of extracting or approximating reward function by observing expert’s behaviour [3]. Similar to RL, IRL is also motivated from behavioural science. To continue with the car driving example, we can think of new-learner. Although it is possible for him to learn driving entirely by himself by trial-error, it could be disastrous as well as time consuming; instead he could learn by observing how the instructor drives and avoiding the common mistakes. Similarly, although ultimately the agent will be deriving its own policy to act; a better reward function can help to arrive at the policy. IRL helps in this concern.

The primary concept and basic mathematical foundation of IRL are laid by [4] which characterizes the problem as Apprenticeship learning. It characterizes entire state space as made up of features *f*. The reward function is defined as the linear combination of these feature set θT*f* i.e. it gives quality values(θ) to each of the features(*f*). The total reward an agent collects through its journey is hence called the feature expectation. E.g. in car driving example, frequent lane changing can be a feature which should be negatively weighted; while on the other hand constant speed could be another one which should be weighted positively. The algorithm assumes that the example trajectories given to it as input are generated by the expert with the goal of maximizing the feature expectation in mind i.e. the policy followed by the expert is optimal or near-optimal. And hence, with the sufficient number of observations in hand we can safely assume that the trajectories not followed by the expert are sub-optimal and should be avoided as they would result in lower feature expectation. With this hypothesis, the algorithm aims to extract the feature weights(θ) so as to maximize the feature expectation as close as possible to that of the experts. The further mathematical calculations then simply rely on solving Linear Programming (LP) equations.

Summarizing all the above points, Inverse Reinforcement Learning can be depicted with the below image.



**Maximum Likelihood (Entropy) Inverse Reinforcement Learning**

The plain IRL algorithm solely depends upon matching feature expectation of apprentice learner with that of trajectory examples demonstrated by expert. The problem with this approach is that many policies and hence many paths may satisfy the feature expectation constraint. Also, it assumes that expert trajectory given as example will always be optimal. But in reality, there could be some noises and sub-optimal behaviour by expert in some cases. To continue with the car driving example, car instructor can also have some glitches in the driving some times, but the leaner is not expected to learn from it and should neglect such noisy behaviour. Maximum Entropy deals with these issue by leveraging probabilistic approach of maximum entropy [10], which allows to extract the policy distribution from expert trajectories which only depends on feature expectation and no other aspects [5]. This approach allows the algorithm to deal with noise and probable suboptimal behaviour in expert trajectories.

The concept is formally defined as below [5],

Notations:

D: demonstrations

M = |D|: number of demonstrations

τ = {s1, a1, s2, a2 . . . st, at}, are the expert trajectories in D, given as input

Rθ= θT *f*T = ∑s ∈ τ (θT *f*s) , reward function defined as linear combination of features ..(1)

It hypothesizes that, the probability of trajectory being followed by the expert is exponentially proportional to its expected reward. By assuming so, it attempts to deal with noise.

i.e. Pr(τ) ∝ e Rθ (τ)

i.e. Pr(τ) = e Rθ (τ)/Z .. (2)

where Z is the normalization term defined as Z= ∑ τ e Rθ (τ)

So now that we have probabilistic model of the trajectory distribution, we can find the parameters (θ) of reward function associated with it by using the mathematical technique of log likelihood i.e. search for θ such that it will maximize the log-likelihood of the probability distribution function (2).

L = argmax θ [ log ∏ τd ∈ D (Pr(τd)) ]

= argmin θ [-(1/M) (∑ τd ∈ D (log (e Rθ (τ)/Z))) ]

(here (1/M) is taken just for the mathematical convenience and it doesn’t affect the optimization problem in hand)

= argmin θ [((1/M) ∑ τd ∈ D  Rθ(τd)) + log ∑ τ e Rθ (τ)]

∇θL = (1/M) (∑ τd ∈ D  ∇θ(Rθ(τd))) – (∑τ Pr(τ) ∇θ(Rθ(τ))

The second term here can be converted to the function of all states S in the trajectories τ, as trajectories are made up of all states.

∇θL = (1/M) (∑ τd ∈ D  ∇θ(Rθ(τd))) – (∑S Pr(S) ∇θ(Rθ(τ))

Using (1),

∇θL = (1/M) (∑ τd ∈ D *f*τd) – (∑S Pr(S) *f*S) .. (3)

Here the first term (∑ τd ∈ D *f*τd) is called feature expectation of the given expert demonstrations i.e. average path features. It is dependent on the trajectories given by expert.

The second term Pr(S) is the state visitation frequency. It is more of environment related property that gives average probability of being in a particular state. It can be calculated using dynamic programming as from expert demonstrations we have probability of a state being start state .

Finally, equation (3) is the optimization equation can now be solved by simply calculating minima using gradient descent following below steps. .. (4)

1. Randomly initialize θ
2. Using Rθ, find current policy π(a|s).

Calculate state visitation frequency Pr(S) using Dynamic Programming

1. Compute gradient in equation (3)
2. Using gradient update θ
3. Go to step 2.

Although these algorithm looks computationally extensive, [6] devised a method to combine it with state-of-the-art deep neural network technique which make the computational complexity independent of the trajectory demonstrations and hence, suitable for the real-world usage.

**2.4. Issues in Inverse Reinforcement Learning**

IRL has been successfully employed in various experimental areas such as Grid-world navigation [11], Car parking [12], Sorting [13] etc. and others. Being an evolving field there have been numerous unresolved ambiguities in it which are shortly summarized below:

1. Generalization

It refers to using limited expert demonstrations given as input to deduce knowledge about the unobserved situations. E.g. start state from where expert never started. The challenge is to generalized Reward function which is not overly fitted to the sample trajectories shown and which generalizes to overall goal of the expert. The IRL algorithm inherently deals with the problem to some extent as it depends on Value function of all states. Another way might be to withhold some of the sample demonstrations as validation set to test the Reward function.

2. Feature selection

Given the IRL algorithm above, where the reward is defined as the function of features; it is straightforward that algorithm’s efficiency is directly associated with how correctly the environment is explained as set of features [14]. It is a way to induce the prior knowledge. The problem becomes challenging when features defined can not model the expert preferences adequately. Sample based approximation methods are suggested to deal with feature selection issue which depend on neural network [17].

3. Multiple Tasks

The expert may demonstrate the related examples with different goals in mind. E.g. driver may drive quiet and steadily when going on an outing; while may drive hastily when going to the office on the next day. In real world scenario, an ideal AI-agent should have capability to deal with such scenarios. [15] [16] make use of probabilistic methods to approach the problem.

4. Nonlinear Rewards

As stated above, reward function is defined as linear combination of features. While this assumption suffices for many problem, it is still inadequate to model features for complex domains such as Surveillance.

3. Design

This chapter high level architecture for the experiments designed to test the IRL algorithms. The first section explains the working components of Maximum Likelihood- Inverse Reinforcement Learning algorithm.

**3.1. Maximum Likelihood- IRL architecture**

The very idea behind this algorithm is to take into consideration the demonstrations shown by expert and generate such a reward function which would maximize the likelihood of given demonstrations. For this purpose, the reward function is defined as the function of states i.e.

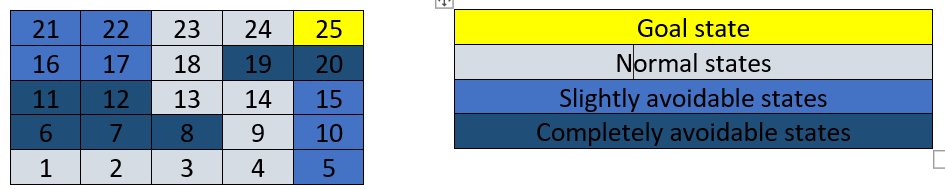
Rθ= θT *f*T (refer 2.3 for details). Here *f*T stands for characteristics features of any state and θis the weightage given to those state features. It can be seen that *f*T representation forms the core part of the algorithm because results would be as good as our ability to differentiate states from each other. The more clearly we differentiate each kind of states from each other, more quickly and correctly the algorithm would work. This is the way to inject background knowledge about environment.

4. Implementation

This chapter introduces specific experiments designed to test the algorithms. The first section details about the simple grid-words environment used to test this algorithm.

**4.1. Grid World environment**

To intuitively understand the working of the algorithm, a simple grid world experiment is performed. A stochastic environment is considered i.e. each action is not guaranteed to result in the same effect each time. Agent’s objective is to reach to goal state avoiding hurdles. The ground reality assumed for the experiment is shown below with characteristics of states. Feature matrix explained above represents these features. To simulate the real-life conditions, we assume that these exact representations may or may not be available to the IRL algorithm. It should cope with such lack of complete knowledge and still work satisfactory. The human knowing which are avoidable states would most often follow the normal path to the goal. But considering the stochasticity of the environment and as expert would not be optimal every time, she may follow the path including the slightly avoidable states too. But she would always avoid completely avoidable states.

****

**Feature Representation:**

The states are represented using two different representations to exhibit complete knowledge of environment and lack of knowledge. All the experiments with grid-world environment are tested using both these extreme cases.

1. Complete knowledge:

The feature vector is given using 4-dimensional vector where each dimension represents each of the characteristics above. This is an ideal situation where you have complete understanding of domain and it states.

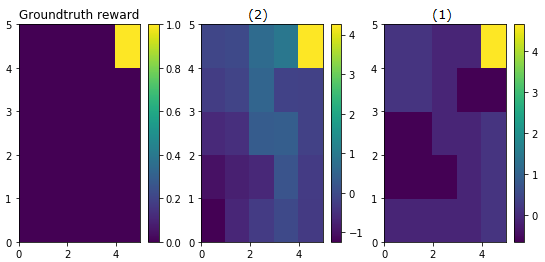
1. Lack of knowledge:

The feature vector is given using 25-dimensional vector where each dimension represents each state differently. This is the worst case where you can not provide any domain knowledge to the system.

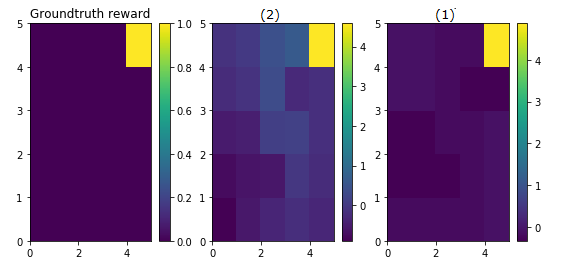
**Experiment 1:**

In the experiment performed, the given demonstrations include 70% of the trajectories following normal states to reach to the goal and remaining 30% of time it would follow slightly avoidable states. For this experiment, all the trajectories ultimately reach to the goal state. The reward recovered after 500 iterations (A) vs. 2000 iterations (B) for different feature representation is illustrated below. The ground truth reward which the expert had in mind while demonstrating is also shown on the left. Figure number (1) & (2) denotes feature representation used as explained above.

A.



B.

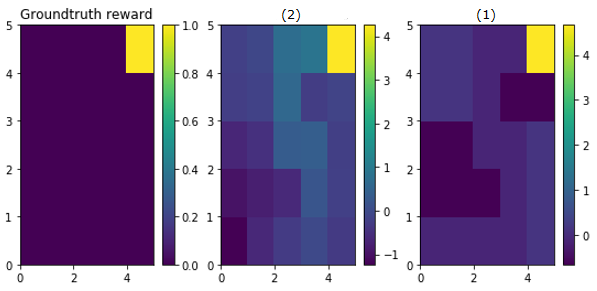


It can be seen from the figure that feature representation (2) i.e. with complete knowledge, performs well which is expected. Also, the performance of the algorithm increases with the number of iterations. Also, even if the feature representation is poor (1), the algorithm slowly moves in correct direction to identify the states’ characteristics. In all the cases goal state is recovered correctly. It can also be seen that algorithm is able to handle the stochasticity and sub-optimality, which is around 30%, of expert demonstrations.

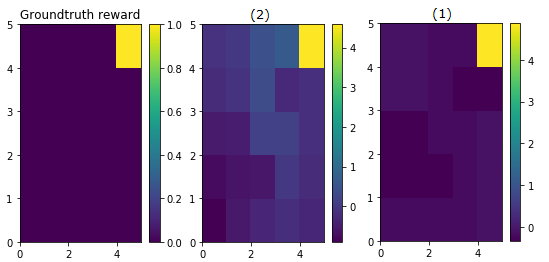
**Experiment 2:**

This experiment’s settings are similar to the first one except input expert trajectories shows more stochastic behaviour; only 33% of trajectories actually reach to the goal while remaining 66% reach to any random state. The results below show the algorithm is still able to recover the goal state satisfactorily.

A.



B.



5. Evaluation

As can be seen from the performed experiments, Maximum Likelihood Inverse Reinforcement Learning algorithm possesses the capacity to recover the reward function is most of the cases, even in cases of sub-optimal behaviour by the expert. These results are in compliance with the mathematical foundation on which the algorithm is based on i.e. trajectories with more reward are exponentially more likely to be demonstrated by the expert. Hence, even in the presence of many random trajectories, few trajectories leading to the goal state could extract the underlying reward function.

One unexpected outcome that emerges from these experiments is related to reward shaping and sparse rewards. Consider the general reinforcement learning where agent is expected to learn the optimal policy by exploring the environment and reward is only provided when and if agent reaches the goal state. Although this appears as a small problem for the above environment like grid world navigation, it has the potential to bring reinforcement learning to crippling speed for complex environments [18]. If we had designed the grid world navigation experiment using reinforcement learning, then agent had to explore the environment many a times to form optimal policy, due to the sparse nature of the reward. As can be seen from recovered reward visuals in the implementation section above, the IRL algorithm solves this problem by assigning different weightage to different states. Hence the reward has been propagated right to the start state and we can actually see the path taken by the expert in reward function itself. This behaviour shows the strength of the algorithm to address sparse reward and cold start problem of reinforcement learning.

**Bibliography**

[1] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning, Second Edition

An Introduction. MIT Press, second edi ed., 2018.

[2] Montague, P. Read et al. “Bee foraging in uncertain environments using predictive hebbian learning.” *Nature* 377 (1995): 725-728.

[3] Ng, Andrew Y. and Stuart J. Russell. “Algorithms for Inverse Reinforcement Learning.” *ICML* (2000).

[4] Abbeel, Pieter and Andrew Y. Ng. “Apprenticeship learning via inverse reinforcement learning.” ICML (2004).

[5] Ziebart, Brian D. et al. “Maximum Entropy Inverse Reinforcement Learning.” *AAAI*(2008).

[6] Wulfmeier, Markus et al. “Maximum Entropy Deep Inverse Reinforcement Learning.” (2015).

[7] Mnih, Volodymyr et al. “Human-level control through deep reinforcement learning.” *Nature* 518 (2015): 529-533.

[8] Mnih, Volodymyr et al. “Playing Atari with Deep Reinforcement Learning.” *CoRR*abs/1312.5602 (2013): n. pag.

[9] Liang, Yitao et al. “State of the Art Control of Atari Games Using Shallow Reinforcement Learning.” *AAMAS* (2016).

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

[11] Fu, Justin et al. “Learning Robust Rewards with Adversarial Inverse Reinforcement Learning.” *CoRR* abs/1710.11248 (2017): n. pag.

[12] Pan, Xinlei et al. “Human-Interactive Subgoal Supervision for Efficient Inverse Reinforcement Learning.” AAMAS (2018).

[13] Bogert, Kenneth D. et al. “Expectation-Maximization for Inverse Reinforcement Learning with Hidden Data.” AAMAS (2016).

[14] Neu, Gergely and Csaba Szepesvári. “Apprenticeship Learning using Inverse Reinforcement Learning and Gradient Methods.” *UAI* (2007).

[15] Babes-Vroman, Monica et al. “Apprenticeship Learning About Multiple Intentions.” *ICML* (2011).

[16] Bogert, Kenneth D. and Prashant Doshi. “Multi-Robot Inverse Reinforcement Learning Under Occlusion with State Transition Estimation.” *AAMAS* (2015).

[17] Finn, Chelsea et al. “Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization.” *ICML* (2016).

[18] Bingyi Kang, Zequn Jie, Jiashi Feng “Policy Optimization with Demonstrations” *ICML* (2018.)

**Dissertation Subject**

Currently, I am finding 3rd issue mentioned above i.e. of multiple task could be the core of the dissertation. I haven’t read enough yet to know if somebody has already done any thorough work in this, but quick recap over the papers available showed that there might still be open areas to work on.

[15] lays foundation for the topic. It uses Expectation-Maximization (EM) technique to first cluster sample trajectories and then find reward weight θ for different intentions. It applies this algorithm in Grid world and Car Driving environment. The possible application area could be personalized home climate control, surveillance etc.

Also, this algorithm needs ‘no. of clusters’ as one of the prior inputs. It can be improved to automatically detect no. of clusters by itself. Another future work the authors suggest is to use this approach to predict behaviour of and better interact with other agents in multi-agent environment.