Model-based Apprenticeship Learning
— Literature Review —

Yuanruo Liang
yuanruo.liang13@imperial.ac.uk

Supervisor: Dr. Marc Deisenroth
Course: CO541, Imperial College London

6th June, 2014

Contents

1 Introduction

In classical reinforcement learning (RL), learning is accomplished within the framework of Markov Decision Processes (MDPs), which provides a model for modelling decision-making where outcomes are partially random and partly under the control of a control agent. In RL, the goal is to obtain an optimal policy which maps states to actions.

In contrast, apprenticeship learning or apprenticeship via inverse reinforcement learning (IRL), where the problem is of extracting a reward function given observed, optimal behavior as demonstrated by an expert, usually assumed to be acting optimally. [1]

In this project, we attempt to use Gaussian Processes, a powerful statistical regression method, as a way to incorporate a learned system model into an IRL algorithm. Additionally, we will attempt to demonstrate the viability of the algorithm via a robotics implementation and evaluating the outcome.

1.1 Accomplishments in the Field

1.1.1 Helicopter Acrobatics

In 2010, Pieter Abbeel, Adam Coates and Andrew Ng successfully presented an apprenticeship learning demonstration of flying helicopter acrobatics, despite the challenging nature of autonomous helicopter flight in general. [2]

Figure 1: Screenshot of helicopter acrobatics. Taken from [2]

1.1.2 Ball-in-a-cup

The ball-in-a-cup is a children's motor game where a ball is hanging from the bottom of the cup via a string. The goal is to toss the ball into the cup by moving only the cup. The actions correspond to the Cartesian accelerations of the cup, and both the state and action space are continuous.

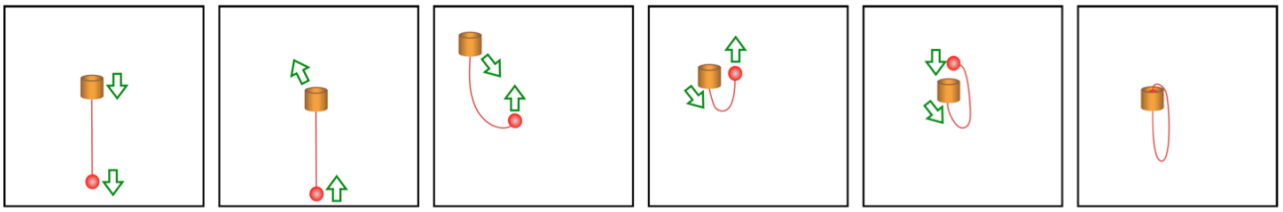

Figure 2: The ball-in-cup problem. The green actions indicate direction of action. Taken from [3]

1.1.3 Cart-pole Swing Up

The cart-pole swing up problem is a physical system where a cart, moving in one dimension, has a stiff pendulum pole attached. The goal is to balance the pendulum above the cart by applying a force

to the cart. The system can be specified by four parameters, $\{x, \dot{x}, \theta, \dot{\theta}\}$ which denote the position and velocities of the cart and the pendulum respectively.

Figure 3: The pole-cart being balanced. Taken from [4]

Using PILCO, the pendulum was balanced with 7 trials and a total experience of only 17.5s. [4]

2 Literature Survey

2.1 Markov Decision Process

2.1.1 Definition

A finite MDP is a tuple $(S, A, P_{sa}, \gamma, R)$ where

1. S is a finite set of N states.
2. $A = a_1, ..., a_k$ is a set of k actions.
3. $P_{sa}(\cdot)$ are the state transition probabilities upon taking action a in state s, i.e. $P_{sa}(s') = P(s'|s, a)$.
4. $\gamma \in [0, 1)$ is the discount factor.
5. $R : S \mapsto \mathbb{R}$ is the reinforcement function with absolute value bound R_{max}.

A policy is defined as any map $\pi : S \mapsto A$. For a given policy π, the value function at a state s_1 is given by

$$V^{\pi}(s_1) = E[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + ...|\pi] \tag{1}$$

which is simply the cumulative expected reward from state s_1 onwards given a policy π, without having chosen any action.

Additionally, the Q-function is defined as

$$Q^{\pi}(s, a) = R(s) + \gamma E[V^{\pi}(s')|P_{sa}] \tag{2}$$

which is essentially the value at a state s after choosing action a, given policy π.

2.1.2 Basic Properties

Let an MDP $M = S, A, P_{sa}, \gamma, R$ and a policy $\pi : S \mapsto A$ be given. Then, two of the classical results concerning finite-state MDPs are [5] [6]

1. Bellman Equations

$$V^{\pi}(s) = \max_{a} Q^{\pi}(s,a) \tag{3}$$

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s')[V^{\pi}(s')] \tag{4}$$

which are essentially Equations (1) and (2) for finite-state MDPs.

2. Bellman Optimality

For an MDP, a policy $\pi : S \mapsto A$ is optimal if and only if for all $s \in S$,

$$\pi(s) \in \arg\max_{a \in A} Q^{\pi}(s,a) \tag{5}$$

Then, for an optimum policy denoted by $*$,

$$V^{*}(s) = \max_{a \in A} Q^{*}(s,a) \tag{6}$$

$$Q^{*}(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s')V^{*}(s') \tag{7}$$

2.2 Algorithms for MDPs

2.2.1 Value Iteration

Here, we assume that an an MDP $M = S, A, P_{sa}, \gamma, R$ be given. The goal is to find the optimal values at every state.

The algorithm used is value iteration:

1. Let $V_0(s) = 0$ for all $s \in S$
2. Update via dynamic programming:

$$V_{k+1} \leftarrow \max_{a} \sum_{s'} P_{sa}(s')[R(s,a,s') + \gamma V_k(s')] \tag{8}$$

The value iteration algorithm always converges from terminal nodes.

2.2.2 Policy Extraction

If, given an MDP, then the best action at every state may be computed using

$$\pi(s) \leftarrow \arg\max_{a} \sum_{s'} P_{sa}(s')[R(s,a,s') + \gamma V^{*}(s')] \tag{9}$$

if the optimal values V* are known, or

$$\pi(s) \leftarrow \arg\max_{a} Q^{*}(s,a) \tag{10}$$

if the optimal q-values Q* are known.

2.2.3 Policy Evaluation

Under a fixed policy, the actions taken are dictated by the policy π, so the value of states can be computed as

$$V^{\pi}(s) = \sum_{s'} P_{s\pi(s)}(s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')] \tag{11}$$

Using Equation (11), we can write the iterative algorithm to evaluate policies for each state $s \in S$:

1. Let $V_0(s) = 0$ for all $s \in S$
2. Update via dynamic programming:

$$V^{\pi}_{k+1}(s) \leftarrow \max_{\pi(s)} \sum_{s'} P_{s\pi(s)}(s')[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')] \tag{12}$$

2.2.4 Policy Iteration

For policy iteration, we assume we have on hand a policy π that is non-optimal, and we wish to improve it. We can combine policy evaluation and policy extraction to improve our policy. This is known as policy iteration:

1. Calculate (non-optimal) utilities for some fixed policy

$$V^{\pi_k}(s) = \sum_{s'} P_{s\pi_k(s)}(s')[R(s, \pi_k(s), s') + \gamma V^{\pi_k}(s')] \tag{13}$$

2. Update policy using policy extraction with converged utility values

$$\pi_{k+1}(s) \leftarrow \arg\max_{a} \sum_{s'} P_{sa}(s')[R(s, a, s') + \gamma V^{\pi_k}(s')] \tag{14}$$

2.2.5 Incorporating GPs into IRL via PILCO

Using Gaussian process regression, it is possible to learn the behavior of a probabilistic model given sufficient measurements of state trajectories and actions.

We will use GP prediction as a method of function approximation, in order to determine the transitive probabilities, which will then be incorporated into IRL and solved for the reward function.

The implementation for this can be easily provided for using PILCO, a practical, data-efficient model-based policy search method. PILCO is able to cope with very little data and can learn with very few trials via approximate inference. [4]

2.3 Algorithms and Techniques in Reinforcement Learning

In reinforcement learning, the learning is still done in an MDP, except that the transition probabilities $P_{sa}(s')$ and the reward/reinforcement function $R(s, a, s')$ unknown. Therefore, in order to discover the transition probabilities and the reward function, it is necessary to interact with the system.

2.3.1 Temporal-Difference Learning

Temporal-difference learning is a model-free method, because R and P are not considered. In temporal-difference learning, a fixed policy π is decided upon. Then, the algorithm is to update a vector of values $V(s)$ each time we experience a sample (s, a, s', r):

1. $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$
2. $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + \alpha \cdot sample$

where α is a learning rate constant. A decreasing learning rate over iterations can give converging averages.

2.3.2 Q-Value Iteration

In temporal-difference learning, the values of the states could be learnt, but it would not be helpful in extracting a new policy without the Q-values. Hence, Q-value iteration:

1. Set $Q_0(s, a) = 0$ for all $s \in S$, $a \in A$
2. Then, do the iterative update

$$Q_{i+1}(s, a) \leftarrow \sum_{s'} P_{sa}(s')[R(s, a, s') + \gamma \max_{a'} Q_i(s', a')] \tag{15}$$

2.3.3 Q-Learning

In Q-learning, the q-values are directly learnt from the samples by iterative updates, similar to temporal-difference learning:

1. $sample = R(s, a, s') + \gamma \max_{\alpha'} Q(s', a')$
2. $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[sample]$

where as before, α is a learning rate constant. A decreasing learning rate over iterations can give converging averages.

2.3.4 Encouraging Exploration

In order to increase the rate of learning, sometimes it is necessary to encourage exploration by choosing actions that lead to unexplored states where the values and/or q-values are unknown.

One method to do this is by using random actions - that is, define a constant $0 < \epsilon < 1$, and thus decide actions randomly with probability ϵ, and non-randomly (by following prescribed policy) with probability $1 - \epsilon$.

However, the problem with this approach is that once all spaces are explored, the agent still thrashes about. Although we can reduce ϵ over time which will solve the problem, we still have to estimate the number of iterations it will take for the agent to visit all states. An alternative approach is to use exploration functions, perhaps one such as:

$$f(u, n) = u + k/n \tag{16}$$

where k is a constant. It can be applied thus:

$$Q_{i+1}(s, a) \leftarrow R(s, a, s') + \gamma \max_{a'} f(Q_i(s', a'), N(s', a')) \tag{17}$$

where $N(s, a)$ is the number of times the agent has taken action a at state s.

By keeping count of the number of times each q-state has been visited in $N(s, a)$, we can set the constant k such that the value of a q-state which is unknown is artificially increased by the exploration functionto encourage exploration in that region.

2.3.5 Feature-based Representations

In realistic situations, it is often impractical to visit every state, because

1. There are too many states to visit them all in training, or
2. Too many states to hold the q-tables in memory, or
3. The state space is continuous

Thus, we would prefer to generalise and learn about only a small number of training states. This is accomplished by using a vector of features to describe the state, rather than the state variables itself. Hence, values and q-values can be represented simply by a weighted set of features:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_d f_d(s) \tag{18}$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + ... + w_d f_d(s,a) \tag{19}$$

With this approach, the experience can be summed up by only a few numbers, which will significantly reduce the state space. However, the disadvantage is that states may share features, but actually have very different values.

2.4 Current IRL Approaches

2.4.1 Dealing with Continuous State Spaces

In a robotics implementation of IRL, the state space of the robot may possibly be infinite.

For instance, the state of a simple straight, stiff robotic arm with a hinge may be described by the state variables θ and $\dot{\theta}$. To ameliorate the problem of intractibility, Andrew Ng and Stuart Russell [1] suggest defining the reward function linearly in the manner

$$R(s) = \alpha_1 \phi_1(s) + \alpha_2 \phi_2(s) + ... + \alpha_d \phi_d(s) \tag{20}$$

where α_is are the parameters to "fit" and ϕ_is are fixed, known basis functions mapping $S \mapsto R$.
The value function following Equation (20) is then given as

$$V^{\pi} = \sum_{i=0}^{d} \alpha_i V_i^{\pi} \tag{21}$$

where V_i^{π} is the expected cumulative reward contributed by $\alpha_i \phi_i(s)$ computed under policy π.

2.4.2 IRL being an ill-defined problem

It has been shown that for R to make the policy $\pi(s) \equiv a_1$ optimal, the condition

$$(P_{a_1} - P_a)(I - \gamma P_{a+1})^{-1} R \succeq 0 \tag{22}$$

must hold for discrete state s, and the equivalent condition

$$E_{s' \sim P_{sa_1}}[V^{\pi}(s')] \geq E_{s' \sim P_{sa}}[V^{\pi}(s')] \tag{23}$$

must hold for continuous states s [1].
However in their paper, they highlighted a few problems, which are

1. Trivial solutions From Equation (22), it is obvious that $R = 0$ (or any other constant vector) is always a solution, because if the reward is the same regardless of action, then any policy, including $\pi(s) \equiv a_1$ will be optimal. Furthermore, there are likely many choices of R that will meet the criteria.

   Without excessive detail, Ng and Russell [1] showed that the full approach to resolve this is to use a linear programming formulation of the problem as such:

$$\text{maximize}_{\alpha} \sum_{i=1}^{N} \min_{a \in \{a_2, ..., a_k\}} \{(P_{a_1}(i) - P_a(i))(I - \gamma P_{a_1})^{-1} R\} - \lambda ||R||_1 \tag{24}$$

   such that $(P_{a_1} - P_a)(I - \gamma P_{a+1})^{-1} R \succeq 0$ for all $a \in A$
   a_1, and $|R_i| \leq R_{max}, i = 1, ..., N$

where $P_a(i)$ denotes the i-th row of P_a and λ is an adjustable penalty coefficient such that R is bounded away from 0 for some constant $\lambda < \lambda_0$.

2. Infinite constraint size

   For large state spaces, there may be infinitely many constraints in the form of Equation (22), which makes it impractical to check them all.; however this can be avoided algorithmically by sampling a large but finite subset of states $S_0 \in S$.

2.4.3 Inability to determine optimal policy in continuous states

By approximating R(s) as a linear function, it may be that it becomes impossibile to express the reward function for which π is optimal. In such a case, the hard constraint given by Equation (22) was modified by introducing a penalty when the constraint was violated. The final linear programming formulation akin to Equation (24) was given by

$$\arg\max_{\alpha} \sum_{s \in S_0} \min_{a \in \{a_2,...,a_k\}} \{p(E_{s' \ P_{sa_1}}[V^{\pi}(s')] - E_{s' \ P_{sa}}[V^{\pi}(s')])\} \tag{25}$$

where

$$p(x) = \begin{cases} x & : x \geq 0 \\ 2x & : x < 0 \end{cases}$$

2.4.4 IRL from Sampled Trajectories

A more realistic case for IRL is when the policy π is known only through a set of actual trajectories in the state space, rather than for the entirety of it.

One method as proposed in [1] is to execute m Monte Carlo trajectories under π. Then, for each basis $i = 1, ..., d$, the value $\hat{V}_i^{\pi}(s_0)$ is defined as the average empirical return on these m trajectories. For instance, if $m = 1$, then

$$\hat{V}_i^{\pi}(s_0) = \phi_i(s_0) + \gamma\phi_i(s_1) + \gamma^2\phi_i(s_2) + ... \tag{26}$$

Then, the estimate for $V^{\pi}(s_0)$ is given as

$$\hat{V}^{\pi}(s_0) = \sum_{i=1}^{d} \alpha_i \hat{V}_i^{\pi}(s_0) \tag{27}$$

3 Problems, Obstacles and Solutions

3.1 Continuous-valued state and action spaces

In real-world applications of MDPs, states and actions are not discrete, but are continuously-valued. Thus there appears to be an infinite set of states that one may be in, and an infinite set of actions to choose from.

One approach, such as by Ziebart et al. [7] use the principle of maximum entropy to probabilistically define a globally normalised distribution over decision sequences.

Another approach by Boularias et al. [3] uses a model-free IRL algorithm, where the relative entropy between the empirical distribution of state-action trajectories under a baseline policy and their distribution under a learned policy is minimised by stochastic gradient descent.

In our approach, we will use gaussian processes via the PILCO framework [4] in order to effectively solve this problem with minimum experience.

3.2 Correspondence problem

The correspondence problem is when the body of the demonstrator (expert) is significantly different from the body of the robot, or when the actions (torques, commands or forces) of the demonstrator are not observed during the actuation.

These drawbacks can be mitigated by using a probabilistic trajectory model approach that is able to reproduce the expert's distribution of demonstrations without the need to observe motor commands during demonstration. [8]

3.3 Computational Cost

During IRL, computing for a full policy in large, continuous state spaces is impractically and computationally expensive. By using a local approximation of the reward function, a method can be used which drops the assumption that the expert demonstrations are globally optimal, requiring only local optimality. [9] This allows for both linear and nonlinear reward functions to be learnt.

3.4 Implementing a suitable robotics demonstration

Live demonstrations of physical IRL systems have so far included demonstrations of pole-balancing and ball-in-a-cup. In this project, a novel robotics demonstration will be thought up and implemented.

Some ideas currently under consideration are:

1. Pushing an item on a table
2. Grappling and lifting an item

4 Progress Summary and Plan

4.1 Present Accomplishments

Firstly, we have obtained robotics support with Dr. Yiannis Demiris, Associate Professor at the Dept. of Electrical and Electronic Engineering. He has given Dr Deisenroth and myself a tour of the laboratory and introduced to us the equipment and robots available. This allows us to formulate a plan of action with the understanding of the available resources and materials.

Additionally, we have reviewed the relevant literature necessary for attempting a robotics implementation of the project.

Furthermore, I have downloaded the PILCO and GPML Toolbox software packages and familiarised myself with it.

Finally, I have attended lectures on Coursera by Andrew Ng, and UC Berkeley by Dan Klein and Pieter Abbeel, which will give me a solid foundation to progress and understand the problems and challenges ahead.

5 Future Phases and End Goal

The final goal of the project is to be able to create a framework which incorporates learned models into inverse reinforcement learning. This should be amply demonstrated with a suitable robotics implementation of the algorithm. A breakdown of the timeline is provided in the Appendix.

References

[1] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *ICML*, pages 663–670, 2000.

[2] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 2010.

[3] Abdeslam Boularis, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning.

[4] Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search.

[5] D. P. Bertsekas and J. Tsitsiklis. Neuro-dynamic programming. *Athena Scientific*, 1996.

[6] R. S. Sutton and A. G. Barto. Reinforcement learning.

[7] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey.

[8] Peter Englert, Alexandros Paraschos, Jan Peters, and Marc Deisenroth. Probabilistic model-based imitation learning.

[9] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples.

A Gantt Chart