UM-DAE CEBS

Reading Project

Course-MPr-801

Reaction network learning for MDP's

Author:
Atthaluri Shashank

Advisor:
Prof. Manoj
GopalKrishnan

ABSTRACT OF THE PROJECT

Reinforcement learning (RL) is a widely known technique to enable autonomous learning.To operate in human-robot coexisting environments, intelligent robots need to simultaneously reason with commonsense knowledge and plan under uncertainty. Markov decision processes (MDPs), are good at planning under uncertainty toward maximizing long-term rewards. Markov decision processes (MDPs) extensions provide an extremely general way to think about how we can act optimally under uncertainty. Here we are going to discuss about what are MDPs. MDP is a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning. Markov decision processes formally describe an environment for reinforcement learning. Where the environment is fully observable. In this project we discuss how to solve the MDPs. And also expose to EM algorithms for reaction network.

Acknowledgements

This project could not have been completed without support and encouragement from these people. My greatest gratitude is extended to:

- My advisor, Manoj, the best mentor I could have. My academic pursuit would have been much harder without the plenty of freedom Manoj gave. The group discussions had helped me learning many things which might not be helpful for these project but it might useful for my future projects.
- The other one is Abhinav. I am grateful to him for teaching me about the EM algorithms and reaction networks and also helped me getting a better big picture of the Reaction networks.

Contents

Chapter 1

1 Introduction

The development of efficient probabilistic inference techniques has made considerable progress in recent years, in particular with respect to exploiting the structure of discrete and continuous problem domains. In this project we show that these techniques can be used also for solving Markov Decision Processes (MDPs). We define policy and value functions. We understand the Bellman optimality equation for solving the MDP. We discuss about what is the EM algorithm and how it can be related to Reaction network.
In this project, we worked on the following things:

- We do a literature review of the Markov Decision Process(MDPs).
- We discuss about the policy an value functions in section 2.2
- We introduce an example of MDP (Recycling Robot)
- We use Bellman optimality equation for solving Recycling robot problem.
- We introduce EM Algorithm and EM algorithm for reaction network.

2 Markov Decision Processes

2.1 Definition 1

In reinforcement learning, an environment is often modeled as a Markov decision process or MDP, where the history of the environment can be summarized in a sufficient statistic called state.
A Markov decision process is defined as a five-tuple:M = $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, where:

- $\mathcal{S}$ is the state space. It can be discrete or continuous
- $\mathcal{A}$ is the action space. Similarly, it can be discrete or continuous.
- T $\in (\mathcal{P}_{\mathcal{S}})^{\mathcal{S}\times\mathcal{A}}$ is a transition function, with $\mathcal{P}_{\mathcal{S}}$ denoting the set of probability distributions over $\mathcal{S}$. If $\mathcal{S}$ is discrete, we may define $T(\cdot \mid s, a)$ for any (s, a) $\in \mathcal{S} \times \mathcal{A}$ as a probability mass function, so that $T(s' \mid s, a)$ is understood to be the probability of reaching a new state s' if action a is executed in state s. If $\mathcal{S}$ is continuous, $T(\cdot \mid s, a)$ is understood to be a probability density function.
- R is a reward function that defines the reinforcements received by the agent. For now on, we will use the word "reward" instead of "reinforcement" as the former is used more often in the literature. In most practical situations, we may assume, without loss of generality, that R is bounded.
- $\gamma \in (0, 1)$ is a discount factor.

Definition 2

Note: An MDP is called finite if both the state and action spaces are finite sets; in contrast, it is called continuous if either the state or action space is continuous. A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment. More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3,.... At each time step t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states, and on that basis selects an action, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t . The agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. Given any state and action s and a, the probability of each possible pair of next state and reward, s', r, is denoted

$$p(s', r \mid s, a) = Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\} \tag{2.1}$$

These quantities completely specify the dynamics of a finite MDP. Most of the theory we present here assumes the environment is a finite MDP.

$$r(s,a) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r \mid s, a) \tag{2.2}$$

the state-transition probabilities,

$$p(s' \mid s, a) = Pr\left\{S_{t+1} = s' \mid S_t = s, A_t = a\right\} = \sum_{r\in\mathcal{R}} p(s', r \mid s, a) \tag{2.3}$$

and the expected rewards for state-action-next-state triples,

$$r(s,a,s') = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r\in\mathcal{R}} rp(s', r \mid s, a)}{p(s' \mid s, a)}. \tag{2.4}$$

2.2 Policy and Value functions

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π , where $\pi(a \mid s)$ is the probability that taken action a when in state s and s $\in \mathcal{S}$, a $\in \mathcal{A}(s)$. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. Policies are also called feedback laws. For any MDP , a stationary policy π determines a distribution of the reward sequence, and thus a distribution of the return. Therefore, we may talk about expected return and use it to evaluate a policy. Naturally, a reward-maximizing agent prefers policies that yield largest expected returns in all states. We define the state-value function, $\upsilon_\pi(s)$, as the expected return by executing π starting from state s

$$\upsilon_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s\right] = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma\upsilon_\pi(s')\right] \tag{2.5}$$

: where $\mathbb{E}_\pi$ refers to the probability distribution of the reward sequence induced by the dynamics of the MDP as well as the policy. Similarly, the state-action value function, $q_\pi(s,a)$, is the expected return by taking action a in states and following π thereafter:

$$q_\pi(s,a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] \tag{2.6}$$

To maximize the total rewards received from the environment, the agent desires an optimal policy π_* whose value functions, denoted by υ_* and q_*, respectively, satisfy the conditions:

$$\upsilon_*(s) = max_\pi \upsilon_\pi(s) \tag{2.7}$$

$$q_*(s) = max_\pi q_\pi(s,a) \tag{2.8}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. For the state-action pair (s, a), this function gives the expected return for taking action a in state s and thereafter following an optimal policy. Thus, we can write q_* in terms of v_* as follows:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right] \tag{2.9}$$

2.3 Value iterations algorithm

Value iteration is probably the simplest and easiest-to-implement algorithm for solving an MDP. This algorithm operates in the value-function space. Starting with an arbitrary, bounded initial value-function estimate, it repeatedly applies the Bellman operator. so that the value-function estimate approaches the optimal value function in the limit. The basic form of value iteration is given in Algorithm 1. For concreteness, the pseudocode initializes the value function to be zero everywhere, but value iteration is guaranteed to converge to the optimal value function for any bounded initial value function. Let V_t be the value-function estimate before the t-th iteration in Algorithm 1. A well-known fact of value iteration is that the sequence of value functions, $[V_t]_{t\in\mathbb{N}}$, approaches V^* at a geometric rate, which follows from the contraction property of the Bellman operator. We define Bellman operator as

$$\mathcal{B}V(s) = max_{a\in\mathcal{A}}\mathcal{B}^a V(s), \forall s \in \mathcal{S} \tag{2.10}$$

where the operator $\mathcal{B}^a$ associated with action a. It should be noted that, even if V_t converges to V^* only in the limit, in practice, we can terminate the algorithm after iteration t when we discover the change of value function, $\parallel V_{t+1} - V_t \parallel_\infty$, drops below a threshold. In fact, it can be shown that the greedy policy π of V_t in value iteration will converge to π^* after finitely many iterations even if V_t may not equal V^* exactly

Algorithm 1 Value iteration

0: **Inputs:** $M = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
1: Initialization: $V(s) \leftarrow 0$ for all $s \in \mathcal{S}$
2: **for** $t = 1, 2, 3, \ldots$ **do**
3: $\quad V \leftarrow \mathfrak{B}V$
4: **end for**

NOTE: Value iterations are repeatedly update an estimate of the optimal value function according to Bellman optimality equation

2.4 An Example of MDP

Recycling Robot

The recycling robot is simple example for MDP. Its basically a robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery. At each time robot has to decide whether it should

- actively searching for cans,
- remain stationary and wait for someone to bring it a can,
- recharge its battery.

The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued which produces a negative reward as a penalty. The agent makes the decisions based on the battery level. So, here states were (high,low) and the actions were (wait, search, recharge). When the energy level is high, recharging would be foolish, so we do not include it in the action set for this state. The agent's action sets are

$$\mathcal{A}(high) = \{search, wait\}, \mathcal{A}(low) = \{search, wait, recharge\} \tag{2.11}$$

The rewards might be zero most of the time, but then become positive when the robot secures an empty can, or large and negative if the battery runs all the way down.If the energy level is high, then a period of active search can always be completed without risk of depleting the battery. A period of searching that begins with a high energy level leaves the energy level high with probability α and reduces it to low with probability 1 - α. On the other hand, a period of searching undertaken when the energy level is low leaves it low with probability β and depletes the battery with probability 1 - β. In the latter case, the robot must be rescued, and the battery is then recharged back to high. Each can collected by the robot counts as a unit reward, whereas a reward of -3 results whenever the robot has to be rescued. Let r_{search} and r_{wait}, with $r_{search} > r_{wait}$, respectively denote the expected number of cans the robot will collect (and hence the expected reward) while searching and while waiting.

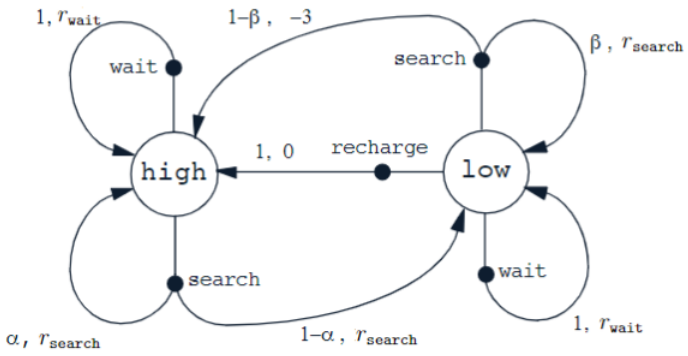
2.5 Solving the above example in finite case

We are going to make the above problem simple, suppose that no cans can be collected during a run home for recharging, and that no cans can be collected on a step in which the battery is depleted. This system is then a finite MDP, and we can write down the transition probabilities and the expected rewards.
For better understanding we drew a transition graph which gives much clarity about the state and how the actions are taken.
There are two kinds of nodes: state nodes and action nodes. There is a state node for each possible state and an action node for each state-action pair.

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

Table: Transition probabilities and expected rewards for the finite MDP of the recycling robot example. There is a row for each possible combination of current state, s, next state, s', and action possible in the current state, $a \in \mathcal{A}(s)$.

Starting in state s and taking action a moves you along the line from state node s to action node (s, a). Then the environment responds with a transition to the next state's node via one of the arrows leaving action node (s, a). Each arrow corresponds to a triple (s, s', a), where s' is the next state, and we label the arrow with the transition probability, $p(s' \mid s, a)$, and the expected reward for that transition, r(s, a, s'). Note that the transition probabilities labeling the arrows leaving an action node always sum to 1.



Bellman Optimality Equations for the Recycling Robot

we give the Bellman optimality equation for the recycling robot. To make things more compact, we abbreviate the states high and low, and the actions search, wait, and recharge respectively by h, l, s, w, and re. Since there are only two states, the Bellman optimality equation consists of two equations. The equation for $v_*(h)$ can be written as follows:

$$\begin{aligned} v_*(\mathtt{h}) &= \max \left\{ \begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{l}) + \gamma v_*(\mathtt{l})], \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{l}) + \gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} \alpha[r_{\mathtt{s}} + \gamma v_*(\mathtt{h})] + (1-\alpha)[r_{\mathtt{s}} + \gamma v_*(\mathtt{l})], \\ 1[r_{\mathtt{w}} + \gamma v_*(\mathtt{h})] + 0[r_{\mathtt{w}} + \gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} r_{\mathtt{s}} + \gamma[\alpha v_*(\mathtt{h}) + (1-\alpha) v_*(\mathtt{l})], \\ r_{\mathtt{w}} + \gamma v_*(\mathtt{h}) \end{array} \right\}. \end{aligned}$$

Following the same procedure for $v_*(\mathtt{l})$ yields the equation

$$v_*(\mathtt{l}) = \max \left\{ \begin{array}{l} \beta r_{\mathtt{s}} - 3(1-\beta) + \gamma[(1-\beta) v_*(\mathtt{h}) + \beta v_*(\mathtt{l})] \\ r_{\mathtt{w}} + \gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array} \right\}.$$

For any choice of r_s, r_w, α, β, and γ, with $0 \leq \gamma < 1$, $0 \leq \alpha, \beta \leq 1$, there is exactly one pair of numbers, $v_*(h)$ and $v_*(\mathtt{l})$, that simultaneously satisfy these two nonlinear equations.

3 An Example Statistical Problem

Consider a three-sided die, say in the shape of a long triangular prism . Here the result is stationary side after the die was rolled. Suppose the probability of

each of the three possible outcomes depends on two hidden parameters θ_1 and θ_2 according to the following statistical model.

$$Pr[\text{“1”}|\theta_1, \theta_2] \propto \theta_1^2 \tag{3.1a}$$
$$Pr[\text{“2”}|\theta_1, \theta_2] \propto \theta_1\theta_2 \tag{3.1b}$$
$$Pr[\text{“3”}|\theta_1, \theta_2] \propto \theta_2^2 \tag{3.1c}$$

Consider an experiment with the die to attempt to discover its statistics. Suppose the die is rolled many times. The fraction of outcomes corresponding to the result “1” is recorded as X_1, and the fractions of outcomes corresponding to the results “2” and “3” are recorded as X_2 and X_3 respectively.

If the three-sided die considered in example (3) is damaged so that the sides “1” and “2” are indistinguishable. In this case, we can only obtain partial observations from experiments. Given the total number of rolls of the die as well as the number of times either ‘1’ or ‘2’ occurred, can we find the parameters which explain the obtained data best? Such a set of parameters is called the **Maximum Likelihood Estimators (MLE)**.

For the above example, it can be mathematically defined as,

$$(\theta_1^*, \theta_2^*) =_{(\theta_1,\theta_2)} Pr[X_1 + X_2, X_3 \mid \theta_1, \theta_2] \tag{3.2}$$

4 A Statistical Solution

The maximum likelihood estimator (2.1.2) can be rewritten as,

$$(\theta_1^*, \theta_2^*) =_{(\theta_1,\theta_2)} \sum_{i=0}^{X_1+X_2} \frac{X_1 + X_2 + X_3}{(i)!\ ((X_1 + X_2) - i)!\ (X_3)!} \theta_1^{(X_1+X_2)+i} \theta_2^{(X_1+X_2+2X_3)-i} \tag{4.1}$$

Consider the case when $(X_1 + X_2, X_3) = (24, 25)$. In this specific problem, maximization with gradient ascent of the likelihood in (4.1), converges to the MLE ratio

$$\frac{\theta_1^*}{\theta_2^*} = \frac{3}{5}$$

4.1 The Expectation Maximization (EM) Algorithm

We can also solve the above problem with an iterative algorithm known as the EM Algorithm. The EM algorithm starts with a guess of parameters and then uses them to sample the values X_1, X_2, X_3 given the partial observation $(X_1 + X_2, X_3)$.

The process of sampling data from parameters is known as the Expectation Projection step or the E-step. After the sampling it finds the maximum likelihood estimators as the full data is obtained in the E-step and updates the new MLE parameters. This step is known as the Mixture Projection step or the M-step. After starting with a guess of parameters, one can iteratively perform the E-M-E-M-E-M... steps. It has not been proven that the algorithm finds the MLE as in (3.2) but in some cases, it converges to the MLE.

5 EM Algorithm Reaction Network (EMCRN)

For the example (3.1), we can do the M-Projection. As the system is log-linear it can be modeled with a 'Design Matrix' below and with the reactions constructed column-wise as described in . The data and the parameters are encoded as concentrations of species of a reaction network.

$$\begin{matrix} X_1 & X_2 & X_3 & \\ \begin{pmatrix} 2 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix} & & & \begin{matrix} \theta_1 \\ \theta_2 \end{matrix} \end{matrix} \tag{5.1}$$

$$\phi \underset{k_4}{\overset{\mathrm{k_1 X_1}}{\rightleftarrows}} 2\theta_1 \tag{5.2a}$$

$$\phi \underset{k_5}{\overset{\mathrm{k_2 X_2}}{\rightleftarrows}} \theta_1 + \theta_2 \tag{5.2b}$$

$$\phi \underset{k_6}{\overset{\mathrm{k_3 X_3}}{\rightleftarrows}} 2\theta_2 \tag{5.2c}$$

The reaction network (5.2) computes the M-Projection for (3.1), given the data as the initial condition encoded as catalysts.
the E-Projection of (3.1), we can create an 'Observation Matrix' as below and with the reactions constructed for each element of a $\mathbb{Z}$-Basis of the kernel of the observation matrix,

$$\begin{matrix} X_1 & X_2 & X_3 & \\ \begin{pmatrix} 1 & 1 & 0 \\ For0 & 0 & 1 \end{pmatrix} & & & \begin{matrix} O_1 \\ O_2 \end{matrix} \end{matrix} \tag{5.3}$$

$$X_1 \underset{k_8\theta_1^2}{\overset{\mathrm{k_7}\theta_1\theta_2}{\rightleftarrows}} X_2 \tag{5.4}$$

The reaction network (5.4) computes the E-Projection for (3.1), given the partial observations as the initial condition and parameters encoded as catalysts.

We can rewrite (5.2) and (5.4).

$$X_1 \xrightarrow{k_1} X_1 + 2\theta_1 \tag{5.5a}$$
$$X_2 \xrightarrow{k_2} X_2 + \theta_1 + \theta_2 \tag{5.5b}$$
$$X_3 \xrightarrow{k_3} X_3 + 2\theta_2 \tag{5.5c}$$
$$2\theta_1 \xrightarrow{k_4} \phi \tag{5.5d}$$
$$\theta_1 + \theta_2 \xrightarrow{k_5} \phi \tag{5.5e}$$
$$2\theta_2 \xrightarrow{k_6} \phi \tag{5.5f}$$
$$X_2 + \theta_1 \xrightarrow{k_7} X_1 + \theta_1 \tag{5.5g}$$
$$X_1 + \theta_2 \xrightarrow{k_8} X_2 + \theta_2 \tag{5.5h}$$

Upon employing mass action kinetics, this formal reaction network (5.5) is shown to performs a continuous time version of the EM Algorithm for (3.1) and converges to a steady state concentration which correspond to the estimate of the MLE, if the rates satisfy

$$\frac{k_1}{k_4} = \frac{k_2}{k_5} = \frac{k_3}{k_6}, k_7 = k_8 \tag{5.6}$$

and the initial concentrations are chosen to be consistent with the experimental partial observations, i.e., if

$$\frac{X_1(0) + X_2(0)}{X_3(0)} = 24/25$$

then

$$\lim_{t\to\infty} \frac{\theta_1(t)}{\theta_2(t)} = \frac{\theta_1^*}{\theta_2^*} = \frac{3}{5}$$

6 Conclusion

In this work, we discussed about what are MDPs and how to solve them. A reinforcement learning problem can be posed in a variety of different ways depending on assumptions about the level of knowledge initially available to the agent. In problems of complete knowledge, the agent has a complete and accurate model of the environment's dynamics. If the environment is an MDP, then such a model consists of the one-step transition probabilities and expected rewards for all states and their allowable actions. In problems of incomplete knowledge, a complete and perfect model of the environment is not available. We also discussed about the EM algorithm and EM algorithm for reaction networks.
In furture we plan to see it is possible to train the MDPs with the existing EM-CRN algorithm as a log linear model. We further show application of MDPs to the Normalized Forward Algorithm, which was done by Abhinav in his Thesis. Symmetrically, it is also possible to implement the Normalized Backward Algorithm.

7 References

[1] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning:An Introduction.

[2] Abhinav, Masters Thesis, Molecular Algorithms and Schemes for their Implementation using DNA.

[3] LIHONG LI, A Unifying Framework For Computational Reinforcement Learning Theory.

[4] Muppirala Viswa Virinchi, Abhishek Behera, Manoj GopalKrishnan, A reaction network scheme which implements the EM algorithm