Markov Decision Process

Markov Decision Process Introductory Tutorial

1 Notation

- s: the state
- a: the action
- $R(s)$: reward associated with state s
- $A \prec B$: the agent prefers A over B
- $A \sim B$: the agent is indifferent with A and B
- $A \preceq B$: the agent prefers A over B or is indifferent with them

2 Markov Decision Process

Markov Decision Process(MDP) is a sequential decision making problem with a fully observable and stochastic environment where the transition model is Markovian with additive rewards.

Fully observable environment is one in which the entire state space is accessible to the agent. In the fully observable environment, the agent does not need memory to make optimal decisions.

Stochastic environment is one in which the action is unreliable. In another word, the action will not uniquely determine the outcome. For example in the game of tossing a coin, you can take the action of tossing the coin but you can not determine the outcome.

A transition model describes the outcomes of each actions in each state. Since the environment is stochastic, we define $Pr(s' \mid s, a)$ as the probability of reaching state s' from state s when action a is done. The Markovian transition model is one that the probability of reaching state s' from state s only depends on state s and the action a but not the history of earlier states.

In each state, the agent will receive a reward which can be either positive or negative, but must be bounded. The reward is the immediate reward the agent received. It is different with the utilities which will be discussed in Section 2.1.

The MDP must have the following parts:

- A set of **State**
- **Transition Model** $Pr(s' \mid s, a)$
- A set of **Action**
- A **Reward** associated with each state

A solution to the MDP is called a *policy* which specifies what the agent should do for any states the agent might reach. The policy is often denoted as π and $\pi(s)$ is the recommended action for state s. Policies are evaluated by the expected reward. The optimal policy is the policy that yields highest expected reward. The optimal policy is often denoted as π^*.

The policy can be either *stationary* or *nonstationary*. In a nonstationary policy, the recommended action for each state can change overtime. For example, in a problem with finite time horizon, the agent only focus on the reward achieved up to some time T. Thus, it is not always optimal for the agent to get to the state with maximal reward. In stationary policy, the recommended action for each state dose not change overtime. For example, in a problem with infinite horizon, there is no reason to change the recommended action for each state overtime since there is no time limit.

2.1 Utility

The agent's preference are captured by the utility function $U(s)$ which is a single number assigned to each state(It is no the same as reward). The expected utility of an action is defined as the sum of the utility of the outcomes weighted by the probability of each outcome. That is:

$$\mathbf{E}\{U(a \mid s)\} = \sum_{s'} Pr(s' \mid s, a)U(s') \tag{1}$$

The principle of *Maximum Expected Utility* says that a agent should choose the action that results in the maximum expected utility. But why choosing the expected utility as the criteria instead of the criteria such as expected squared utility? This question can be answered by examining the properties of the preferences a rational agent should have. To this end, we can think of the set of outcomes of each action as a $lottery = [p_1, S_1; p_2, S_2; ...; p_n, S_n]$ where the outcomes are $S_1, S_2, ..., S_n$ and each of them happen with probability $p_1, p_2, ..., p_n$.

- *Orderability*: Given any two lotteries, the rational agent must either prefer one of them or consider them as equally preferable.

$$\text{Exactly one of } A \prec B,\ A \sim B \text{ or } A \preceq B \text{ holds}$$

- *Transitivity*: Given three lotteries. If the agent prefers A over B and prefers B over C. Then the agent must prefers A to C.

$$A \prec B \text{ and } B \prec C \Rightarrow A \prec C$$

- *Continuity*: Given three lotteries. If B is between A and C in preference. Then there exists a probability p such that the new lottery which takes A with probability p and C with probability $1 - p$ will be the same as B in preference.

$$A \prec B \prec C \Rightarrow \exists p\ [p, A; 1 - p, C] \sim B$$

- *Substitutability*: Given two lotteries. If the agent has preference between A and B, then the preference still hold between two more complex lotteries that generated from substituting the same part of A and B with C.

$$A \sim B \Rightarrow \forall p\ [p, A; 1 - p, C] \sim [p, B; 1 - p, C]$$

This also hold when substitute $\sim$ with $\prec$ or $\preceq$.

- *Monotonicity*: Suppose two lotteries have the same two outcomes A and B. If the agent prefers A over B, then the agent must prefer the lottery with a higher probability of A.

$$A \prec B \Rightarrow (p > q \ [p, A; 1-p, B] \prec [q, A; 1-q, B])$$

- *Decomposability*: Compound lotteries can be decomposed into a simpler one using the law of probability.

$$[p, A; (1-p)[q, B; 1-q, C]] \sim [p, A; q(1-p), B; (1-p)(1-q), C]$$

From the above six axioms of utility, we can derive the following consequences (see von Neumann and Morgenstern, 1944 for proof)

- *Existence of Utility Function*: There exists a function U such that $U(A) > U(B)$ if and only if $A \prec B$, and $U(A) = U(B)$ if and only if $A \sim B$.
- *Expected Utility of a Lottery*: The utility of a lottery is the sum of the utilities of the outcomes weighted by their probabilities.

$$U([p_1, S_1; ...; p_n, S_n]) = \sum_i p_i U(S_i)$$

2.2 Utility Function

There are four types of MDP based on the difference in time horizon and the criteria:

1. *Finite Time Horizon T*: The problem is characterized by a deadline at time T. In this case, the agent only focus on the reward achieved up to time T.
2. *Infinite Time Horizon With Terminal State*: The problem never terminate because of time. But the agent will eventually reach a terminal state.
3. *Infinite Time Horizon with Discount*: The problems never end but the rewards are discounted by a factor γ.[1] In this case, the rewards from a closer time are more important.
4. *Infinite Time Horizon With Average Reward*: The problem never terminate but the agent focuses on the average reward.

The Bellman function for different time horizons are different. We have

- *Finite Time Horizon T*:

$$U_t(s) = R(s) + \max_a \left\{ \sum_{s'} Pr(s \mid s', a) U_{t+1}(s') \right\} \tag{2}$$

The utility function for each state is time sensitive.

[1] The existence of discount factor γ guarantees that to total reward achieved is finite. Since

$$R[S_1, S_2, ...] = \sum_{t=0}^{+\infty} \gamma^t R(S_t) \leq \sum_{t=0}^{+\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$$

- *Infinite Time Horizon With Terminal State*:

$$U(s) = R(s) + \max_a \left\{ \sum_{s'} Pr(s \mid s', a) U(s') \right\} \tag{3}$$

- *Infinite Time Horizon with Discount*:

$$U(s) = R(s) + \max_a \left\{ \gamma \sum_{s'} Pr(s \mid s', a) U(s') \right\} \tag{4}$$

  where $\gamma \in (0, 1]$ is the discount factor

- *Infinite Time Horizon With Average Reward*:

$$\theta + U(s) = R(s) + \max_a \left\{ \sum_{s'} Pr(s \mid s', a) U(s') \right\} \tag{5}$$

  where θ is the optimal average reward.

3 Optimal Policy

Now, the remaining problem is how to get the optimal policy. Next, we will show two popular algorithms that can be used to get the optimal policy.

3.1 Value Iteration

The basic idea of *Value Iteration Algorithm* is to calculate the utility of each state and then use the utility to choose the optimal action at each state. If there are N states, we will have N Bellman equations with N unknown variables. The problem is that the Bellman equation is not linear because of the *max* operation. One possible way is *iterative*. We start with arbitrary initial values and calculate the right-hand side of bellman equation. Then, we use the result to update the left-hand side of the bellman equation. We repeat the process until the values converge. Let the utility for state s at iteration t be $U_t(s)$. The general update rule of value iteration algorithm looks like:

$$U_{t+1}(s) = R(s) + \max_a \left\{ \gamma \sum_{s'} Pr(s \mid s', a) U_t(s') \right\} \tag{6}$$

The value iteration algorithm is guaranteed to converge to the solution of the Bellman equation when the number of iteration goes to infinity. When only updating s subset of the states at each iteration, the algorithm becomes *Asynchronous Value Iteration*.

3.1.1 Convergence of Value Iteration

The proof of the convergence involves the notion of *Contraction*. The contraction here is a function with one argument that when applying to two inputs, it produces two outputs that the difference is closer by

some constant factor than the difference of the input. For example, the function $f(x) = \frac{x}{2}$ is a contraction with the constant factor equals to 2. A contraction has only one fixed point. When applying the function to any input, the output will get closer to the fixed point. We define an operator $||U|| = \max_s |U(s)|$ which is the maximum of the absolute value of elements. Then $||U - U'|| = \max_s |U(s) - U'(s)|$ which is the maximum difference between any two elements s. The proof is based on the following result

$$||BU_i - BU_i^{'}|| \leq \gamma ||U_i - U_i^{'}|| \tag{7}$$

where B is the Bellman operator. BU is equivalent to applying the Bellman equation that is

$$BU(s) = R(s) + \max_a \{\gamma \textstyle\sum_{s'} Pr(s' \mid s, a) U(s')\}$$

The optimal utility function U is the fixed point of Bellman operator, we have $BU = U$, when applying it to (7), we have

$$||BU_i - U|| \leq \gamma ||U_i - U|| \tag{8}$$

Then we can see that the algorithm converges exponentially fast. Since the utility of all the states are bounded by $\pm R_{max}/(1-\gamma)$ and use the $||U_i - U||$ as the error of U_i, we have the maximum initial error $||U_i - U_0|| \leq 2R_{max}/(1-\gamma)$. Suppose after N iteration, the error is at most ϵ. Then we have $\gamma^N \cdot 2R_{max}/(1-\gamma) \leq \epsilon$. After some rearrangement, we have

$$N = \lceil log(2R_{max}/\epsilon(1-\gamma))/log(1/\gamma) \rceil \tag{9}$$

Next, we will prove that (7) holds.

$$\begin{aligned}
||BU_i - U|| &= \max_s |BU_i(s) - BU(s)| \\
&= \max_s \gamma \cdot | \max_a \sum_{s'} Pr(s' \mid s, a) U_i(s') - \max_a \sum_{s'} Pr(s' \mid s, a) U(s') \mid \\
&\leq \max_s \gamma \cdot \max_a | \sum_{s'} Pr(s' \mid s, a) U_i(s') - \sum_{s'} Pr(s' \mid s, a) U(s') \mid \\
&= \max_s \gamma \cdot \max_a \sum_{s'} Pr(s' \mid s, a) \mid U_i(s') - U(s') | \\
&\leq \gamma \cdot \max_s |U_i(s) - U(s)| \\
&= \gamma \cdot ||U_i - U||
\end{aligned} \tag{10}$$

Line 3 holds because $|\max_x f_1(x) - \max_x f_2(x)| \leq \max_x |f_1(x) - f_2(x)|$. The penultimate line holds because $\sum_{s'} Pr(s' \mid s, a) = 1$.

3.2 Policy Iteration

We can observe that the agent only care about the rank of states in preference. The actual magnitude of the utility need not to be precise. In an other world, the agent can get a optimal policy when the utility is not accurate. This insight suggests an alternative algorithm to get the optimal policy that is *Policy Iterate Algorithm*. The policy iteration algorithm involves the following two steps:

1. *Policy Evaluation*: Given a policy π_i, calculate the utility of each state when policy π_i is executed.

2. *Policy Improvement*: Get the updated policy π_{i+1} using the following rule:

$$\text{when } \max_a \sum_{s'} Pr(s' \mid s,a)U^{\pi_i}(s) > \sum_{s'} Pr(s' \mid s,a)U^{\pi_i}(s)$$

$$\pi_{i+1}(s) = \operatorname*{argmax}_a \sum_{s'} Pr(s' \mid s,a)U^{\pi_i}(s) \tag{11}$$

The algorithm requires checking for update for each state at one iteration. It turns out that it is not necessary. At each iteration, you can choose a subset of the state to perform the policy improvement or the value iteration. This method is called *Asynchronous Policy Iteration*. This algorithm is also guaranteed to converge to the optimal policy with certain initialization.

4 Reference

[1] Russell, Stuart J. (Stuart Jonathan). Artificial Intelligence : a Modern Approach. Upper Saddle River, N.J. :Prentice Hall, 2010.

[2] `http://chercheurs.lille.inria.fr/~lazaric/Webpage/MVA-RL_Course13_files/notes-lecture-02.pdf`

[3] `https://homes.cs.washington.edu/~todorov/courses/amath579/MDP.pdf`

[4] `http://www.cs.cmu.edu/afs/cs/academic/class/15780-s16/www/slides/mdps.pdf`