

Chapter 3

Thursday, January 3, 2019 1:15 PM

Finite Markov Decision Processes (Finite MDP):

The Finite MDP problem involves both evaluative feedback and also an associative aspect (choosing different actions in different situations). They are a classical formalization of sequential decision making, where actions influence not only the immediate rewards but also subsequent situations, or states and through those future rewards also. So they involve delayed reward and the need to tradeoff between immediate and delayed reward.

In the bandit problem only the value of each action ($q^*(a)$) is estimated but in MDPs, the value $q^*(s,a)$ of each action in each state 's' has to be estimated or the value $v^*(s)$ of each state given optimal actions selection have to be estimated. These state dependent quantities are essential to accurately assign credit for long-term consequences to individual action selections.

3.1. The Agent-Environment Interface

MDPs are a straight forward way to frame the problem of learning from interaction to achieve a goal.

Agent:

The learner and decision maker.

Environment:

The thing the agent interacts with, it comprises everything outside the agent.

Interactions:

The agent and environment interacts continually. The agent selects actions and the environment responds to those actions and present new situations to the agent. The environment also provides the rewards, which are nothing but the numerical values that the agent has to maximize over time through its choice of actions.

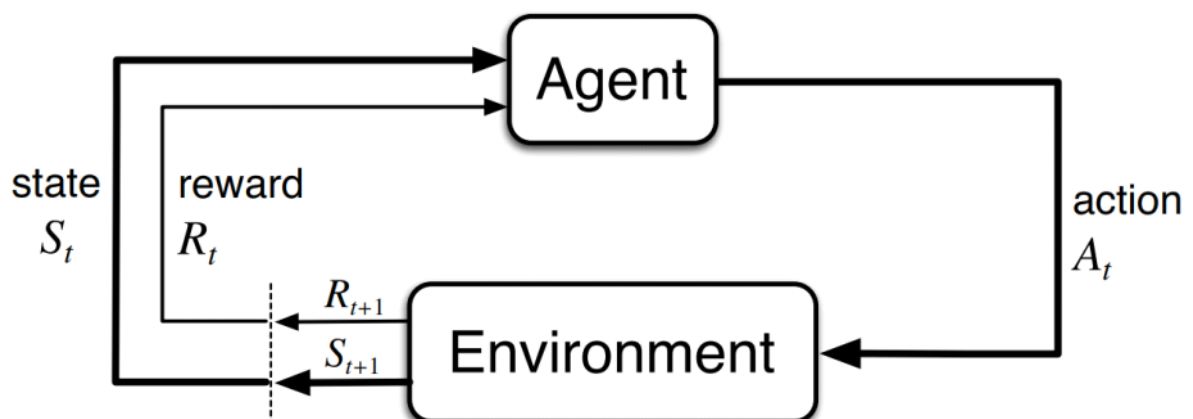


Figure Description: The agent-environment interaction in a Markov Decision Process

The agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \dots$. At each time step t , the agent receives some representation of the environment's state, $S_t \in S$, and on that basis selects an action, $A_t \in A(s)$. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in R$, and finds itself in a new state, S_{t+1} .

The MDP and agent together thereby give rise to a sequence or *trajectory* that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

In finite MDP, the sets of states, actions and rewards (\mathcal{S} , \mathcal{A} , and \mathcal{R}) have a finite number of elements. The random variables R_t and S_t have well defined discrete probability distributions which depend only on the previous state and the action. For particular values of the random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at time t , given particular values of the preceding state and action:

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

This function 'p' defines the dynamics of MDP. 'p' specifies a probability distribution for each choice of s and a , that is, that

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

The probabilities given by 'p' completely characterize the environment's dynamics. So the probability of each possible value for S_t and R_t depend only on the immediate preceding state and action, S_{t-1} and A_{t-1} not on the earlier states and actions.

This is a restriction to the states and this restriction implies that state must include information about all aspects of the past agent-environment interaction that make a difference for the future. It is called as the "Markov Property".

The four argument dynamics function 'p' can be used to compute anything about the environment. The state transition probabilities can be found out using the following relation:

$$p(s' | s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

which becomes a three argument function.

$$p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$$

The expected rewards can also be computed for the state action pairs using the following equation and this is a two argument function.

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

The two - argument function:

$$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

The expected rewards for the state-action-next-state triples as a three-argument function.

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

The three argument function:

$$r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$$

The MDP framework is abstract and flexible. They can be applied to different problems in different ways.

Example Considerations:

Example 1: (Time steps)

The time steps need not to refer to the fixed intervals of real time. They can refer to arbitrary successive stages of decision making and acting.

Example 2: (Actions)

The actions can be low-level controls, such as voltages applied to motors of robot arm or high-level decisions such as whether or not to have lunch or to go to graduate school.

Example 3: (States)

States can be completely determined by low-level sensations, such as direct sensor readings, or they can be more high-level and abstract, such as symbolic description of objects in a room. Some of what makes up a state can be based on memory of past sensations or even be entirely mental or subjective.

Broader Considerations:

An agent can be in the state of not being sure where an object is or of having just been surprised in some clearly defined sense.

Similarly some actions might be totally mental or computational. Some actions might control what an agent chooses to think about or where it focuses its attention.

In general, actions can be any decisions that have to be learned how to make, and the states can be anything to know which might be useful in making them.

Boundary between the agent and the environment:

Boundary between the agent and the environment is not necessarily to be same as the physical boundary of a robot's or animal's body.

For example, the motors and mechanical linkages of a robot and its sensing hardware should be considered as part of the environment rather than parts of the agent. If the MDP is applied to a human or animal muscles, skeleton, and sensory organs should be considered part of the environment.

Rewards may be calculated inside the physical bodies of natural and artificial learning systems, but should be considered external to the agent.

So the general rule is, anything that cannot be changes arbitrarily by the agent is considered as outside of it and belongs to the part of its environment.

It is not necessary to consider everything in the environment is unknown to the agent. The agent often knows quite a bit about how the rewards are computed as a function of its actions and the states in which they are taken. But the reward computation is considered as external to the agent, because it defines the task facing the agent and beyond the agent's ability to change arbitrarily.

For example, in a complicated robot, many different agents may be operating at once, each with its own boundary. One agent may take high-level decisions which form part of the states faced by a lower-agent that implements the high-level decisions.

Agent - Environment boundary is selected once particular states, actions and rewards were selected and also specific decision making task of interest is identified.

The MDP framework suggests that any problem of goal-directed learning can be reduced to three signals passing back and forth between an agent and its environment:

1. One signal to represent the choices made by the agent (actions)
2. One signal to represent the basis on which choices are made (state)
3. One signal to define the agent's goal (reward)

The particular states and actions vary greatly from task to task and how they are represented can

strongly affect the performance. Such representational choices are more art than science.

3.2. Goals and Rewards:

In RL, purpose / goal of agent is denoted by a special signal called reward, passed by environment to the agent.

At each timestep, the reward is a simple number $R_t \in \mathbb{R}$.

The agent's goal is to maximize the total amount of reward. So the agent has to maximize not only the immediate reward but also the cumulative reward in the long run.

So the "reward hypothesis" can be stated as follows:

That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The use of reward signal to formalize the idea of a goal is one of the most distinctive features of RL.

Example cases of reward assignment:

Walking robot:

To make a robot to learn to walk, reward will be provided on each time proportional to the robot's forward motion.

Maze solving agent:

The reward is often -1 or every time step that passes prior to escape, which encourages the agent to escape as quickly as possible.

Empty Soda Can Collecting Agent:

Reward of zero for most of the time and +1 for each can collected. Give negative rewards when the robot bumps into things or when somebody yells at it.

Checkers / Chess Playing Agent:

The natural rewards are +1 for winning, -1 for losing, and 0 for drawing and for all nonterminal positions.

So if we want the agent to do something, the rewards have to be provided in such a way that in maximizing them the agent will also achieve the goals. Reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want to do.

This can be visualized using the following example:

A chess playing agent should be rewarded only for actually winning, not for achieving sub goals such as taking its opponent's pieces or gaining control of the center of the board. If achieving these sorts of sub-goals were rewarded, then the agent might find a way to achieve them without achieving the real goal. For example, it might find a way to take the opponent's pieces even at the cost of losing the game. *The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved*

3.3. Returns and Episodes:

The agent has to maximize the *expected return*. The return can be denoted as G_t and it is defined as some function of the reward sequence. The return is the sum of rewards.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

where T is the final time step. This approach is suitable for the applications in which there is a final step. In these applications the agent-environment interaction breaks naturally into subsequences, called episodes. Some of the examples of episode are plays of a game, trips through a maze, or any sort of repeated interaction.

Each episode ends in a special state called the terminal state and then reset to a standard starting state or to a sample from standard distribution of starting states. Each new episode begins independently of

how the previous episode ended. Episodes can be considered to end in the same terminal state with different rewards for different outcomes. Tasks with episode of this kind are called as *episodic tasks*. It is needed to distinguish the set of all nonterminal states (S) from the set of all states plus the terminal state (S⁺). The time of termination (T) is a random variable that normally varies from episode to episode.

In many-cases the agent-environment interaction does not break into identifiable episodes, but goes on continually without limit. These are called as *continuing tasks*. It is the natural way to formulate an on-going process-control task or an application to a robot with a long life span. The return formulation for the continuing tasks is challenging because the final time step would be $T = \infty$, and the return which the agent tries to maximize will also be infinite.

Concept of Discounting:

The agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. So, the agent chooses A_t to maximize the expected discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $0 \leq \gamma \leq 1$ is called as the discount rate.

The *discount rate* determines the present value of future rewards. Because the reward received k time steps into the future is worth only γ^{k-1} times what it would be worth if it was received immediately.

If $\gamma < 1$, the infinite sum in the above equation has a finite value as long as the reward sequence $\{R_k\}$ is bounded.

If $\gamma = 0$, then the agent is concerned only with maximizing the immediate rewards, this agent is "myopic" (short-sighted). It's objective is only to learn how to choose A_t so as to maximize only R_{t+1} . Acting to maximize the immediate reward can reduce access to future rewards so that the return is reduced. As γ approaches 1, the return objective takes future rewards into account more strongly and hence the agent becomes far sighted.

Return at each time steps are related to each other in a way which is important for the theory and algorithms of reinforcement learning.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

The above relationship works for all time steps $t < T$, even if the termination occurs at $t+1$, if $G_T = 0$. This formula makes it easy to compute the returns (a form of incremental computation).

Although the return is a sum of an infinite number of terms, it is still finite if the reward is nonzero and constant if $\gamma < 1$.

For example if the reward is a constant +1, then the return can be considered as below:

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma}$$

3.4 Unified Notation for Episodic and Continuing Tasks

It is essential to develop common notations to discuss about both the episodic and the continuing tasks.

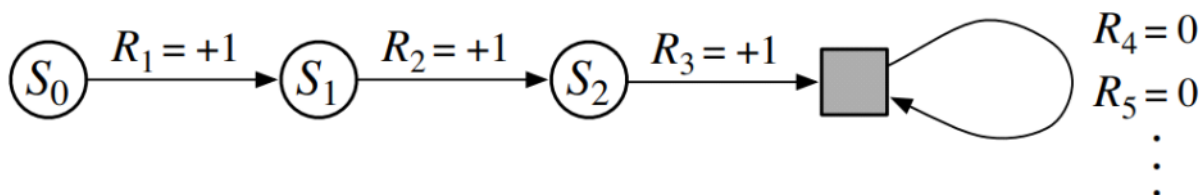
Episodic tasks require some additional notation. It is needed to consider a series of episodes, each of which contains finite sequence of time steps. So the state representation has to be given in the form of $S_{t,i}$ which represents the state at time 't' and episode 'i'. This has to be followed for the other terms like $A_{t,i}$, $R_{t,i}$, $\Pi_{t,i}$, T_i . During discussion of episodic tasks it is not needed to distinguish between different episodes.

Either a particular single episode is considered or something that is true for all episodes is stated.

So there is a need to represent the "return" in both the episodic and continuing tasks.

In episodic tasks the return is a sum over a finite number of terms and in continuing tasks it is a sum over infinite number of terms. These two can be unified by considering the episode termination to be the entering of a special absorbing state that transitions only to itself and that generates only rewards of zero.

The following example states the same consideration discussed above.



The solid square represents the end of an episode (which is special absorbing state).

So starting from S_0 , the reward sequence is +1,+1,+1,0,0,... So when these are added it same as that of the first T reward (here T is 3) or over the full infinite sequence.

It is also true in case of discounting. So the return can be defined in general using the convention of omitting the episode numbers when they are not needed and including the possibility that $\gamma = 1$ if the sums remain defined (all the episodes terminate). So the return can be written as follows:

$$G_t \doteq \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

Including the possibility that $T = \infty$ or $\gamma = 1$, but not the both.

3.5. Policies and Value Functions:

Value Functions:

RL algorithms involve estimating the value functions. These are the functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good is to perform a given action in a given state).

So the notion "how good" is defined as the future rewards that can be expected or the expected return. Value functions are also defined with respect to particular ways of acting, called policies.

Policies:

A policy is a mapping from states to probabilities of selecting each possible action.

Π is an ordinary function; the " Π " in the middle of $\Pi(a|s)$ merely reminds that it defines a probability distribution over $A \in A(s)$ for each $S \in S$.

The value function of a state 's' under a policy ' Π ' denoted as $V_{\Pi}(S)$, is the expected return when starting

in 's' and following ' Π ' thereafter.

For MDPs $V_{\Pi}(S)$ can be denoted as follows.

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S},$$

The function $V_{\Pi}(S)$ is called as the state-value function for policy Π .

The value of taking action 'a' in state 's' under policy ' Π ', $q_{\Pi}(s,a)$ can be denoted using the following equation.

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

$q_{\Pi}(s,a)$ is called as the action-value function for policy Π .

Both the state value function and action value function can be estimated from the experience.

If an agent follows policy Π and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $v_{\Pi}(s)$, as the number of times that state is encountered approaches infinity.

Same way if separate averages are kept for each action taken in each state, then these averages will similarly converge to the action values, $q_{\Pi}(s,a)$.

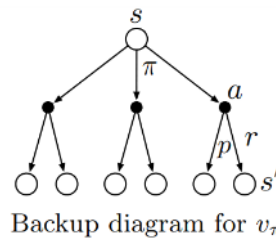
The estimation methods of this type are called as Monte-Carlo methods. They involve averaging over any random samples of actual returns.

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy recursive relationships.

$$\begin{aligned} v_{\pi}(s) &\doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s'] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_{\pi}(s') \right], \quad \text{for all } s \in \mathcal{S}, \end{aligned}$$

where it is implicit that the actions, a , are taken from the set $A(s)$, that the next states, s_0 , are taken from the set \mathcal{S} (or from \mathcal{S}^+ in the case of an episodic problem), and that the rewards, r , are taken from the set \mathcal{R} .

The above equation is called the **Bellman Equation for v_{Π}** . This equation describes the relationship between the value of a state and the values of its successor states. The following diagram actually represents that relationship.



Description about the previous example:

Each open circle represents a state and each solid circle represents a state-action pair. Starting from the root node 's' which is a state at the top, the agent can take any of some set of actions (three) based on the policy Π .

The environment could then respond with one of several next states, s' (two), along with a reward 'r', depending on the dynamics given by the function 'p'.

The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. So, the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

The backup diagrams form the basis of the backup operations which are the heart of RL.

These backup operations transfer the value information back to a state(or a state action pair) from its successor state (or successor state action pairs).