MACHINE INTELLIGENCE COMMUNITY

Summary of Reinforcement Learning: An Introduction Chapter 3: Finite Markov Decision Processes

Justin Chen* chenjus@bu.edu

Tyrone Hou* tyroneh@bu.edu

January 15, 2018

Introduction

A **Markov Decision Process (MDP)** is a mathematical formalization for the reinforcement learning problem.

Characteristics

- **Evaluative feedback** (reward signal *how good* each action is)
- **Association** (the reward for an action also depends on the current state)
- Future states depend on present actions
- **Delayed reward** (rewards may not be received until many time steps have passed)
- Value functions $v_*(s)$ are functions that estimate expected reward of states *and* actions $q_*(s, a)$
  - Compared to bandit problems which modeled expected reward given an action $q_*(a)$

3.1 The Agent-Environment Interface

- **Agent** - learner and decision maker
- **Environment** - system that the agent interacts with and receives reward signals from
- Agent and environment interact over time and each time step:
  - Agent observes state $S_t \in \mathcal{S}$
  - Chooses available action $A_t \in \mathcal{A}(s)$
  - Receives a **numerical reward** at next time step, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
  - S_t, A_t, and R_t are random variables
- **Trajectory** - repeated sequence of state, action, reward:

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

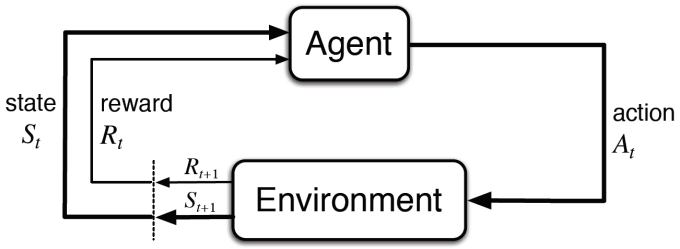
- In finite MDPs, $\mathcal{S}, \mathcal{A}, \mathcal{R}$ are all discrete sets (finite number of elements)
- Each state $s \in \mathcal{S}$ and reward $r \in \mathcal{R}$ has an associated probability of occurring at time t given previous state and action described by this definition:

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

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

*These authors contributed equally to this work

Figure 1: agent-environment interaction in a Markov decision process



- Four argument function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ fully characterizes dynamics of the finite MDP (how it changes over time) and can be used to query environment:
  - $s' \in \mathcal{S}$ (next state)
  - $r \in \mathcal{R}$ (next reward)
  - $s \in \mathcal{S}$ (current state)
  - $a \in \mathcal{A}$ (current action)

  For example a finite MDP can be used to compute the following functions:
  - **State-transition probabilities**

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

  - **Expected rewards for state-action pairs**

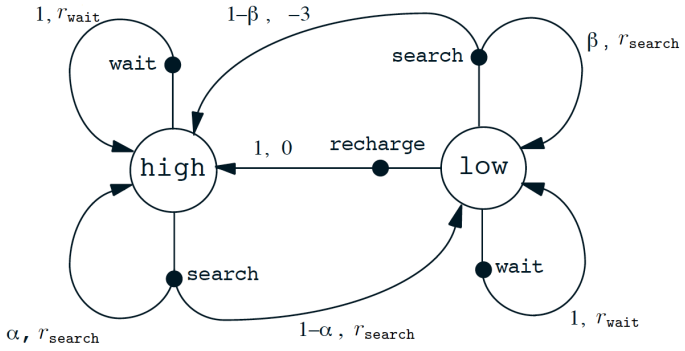
$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

  - **Expected rewards for state-action-next-state triples**

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

- **Agent-environment Boundary**
  - Anything that cannot be changed arbitrarily by the agent is considered to be part of the environment
  - Usually assume that reward is computed outside of the agent and it cannot be changed arbitrarily
  - Think of the boundary as a way to abstract tasks and you can compose these tasks - these components - to form larger components that collectively solve a larger task
  - Boundary is determined once one has selected particular states, actions, and rewards, and thus has identified a specific decision making task of interest
- **Transition Graph**
  - Summarizes dynamics of MDP
  - *State node* for each possible state s (large open circle labeled with name of state)
  - *Action node* on each arrow for each state-action pair (s, a) (small solid circle labeled with name of action)
  - Arrows represent transition triple (s, s', a) where s' represents the next state
    - Labeled with transition probability $p(s'|s, a)$ and expected reward $r(s, a, s')$

Figure 2: State transition example



3.2 Goals and Rewards

- Agent's goal is to maximize long-term expected reward
- Agent receives reward $R_t \in \mathbb{R}$ at each time step
- **Reward Hypothesis** - All goals and motivation can be considered as maximizing long-term cumulative sum of rewards
- Rewards should indicate the quality of behavior that is conducive to solving a particular task
  - Think of grades. Points are arbitrarily allocated to impose the professor's preferences. Misallocation of points can lead to encouraging incorrect tasks such as optimizing for the reward - solving a subtask - instead of optimizing for the goal - learning the material.
  - Further consider unbalanced rewards and again consider grades. Allocating more points to the final exam indicates that the each question on the final example is more valuable than the content considered in prior exams. This can skew the true value of concepts that should be learned and the main objective of the course. To further highlight the importance of properly allocating rewards, over representing or under representing subtasks (exam questions or topics during lecture) can mislead agents (students) from learning the overall goal of the course (environment).
  - Grades can be arbitrary and may not properly reflect the quality of understanding. They reflect that points were arbitrarily allocated or that the instructor itself does not have a strong enough understanding of the material to properly balance rewards for concepts.
  - Reward signals communicate what you want to achieve, not how it should be achieved
    - Imparting how things should be achieved should be constructed in the initial policy or initial value function

3.3 Returns and Episodes

- **Return** - Sum of rewards, denoted $G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T$
- **Episode** - a repeated subsequence of the action-environment interaction
  - e.g. A lap around a track
  - **Terminal State** - Special state that each episode ends in
  - Episodes resume to a starting state after the terminal state independent of all previous episodes
  - This approach should be applied when the sequence can be logically decomposed into episodes and there is a final time step
  - Agent's goal is to maximize its **expected return**
  - **Episodic Tasks** - The set of possible outcomes with distinct rewards for an episode that end in the same terminal state

- **Continuing Tasks** - An agent-environment interaction that does not logically decompose into episodes but instead continues indefinitely
  - e.g. Global human activity (here, agents are humans and actions are human activity) and climate system (environment) is a continuous sequence of agent-environment interaction that does not decompose into episodes. One may consider that days could represent logical episodes, but in reality human activity never stops. There are always people or human-made systems (power plants, etc.) somewhere in the world that are actively interacting with the environment. Everything does not simultaneously start and restart, which is required to define an episode.
  - Here, the final step $T = \infty$ implying that the return can also be infinite
- **Discounting** - Action selection such that the sum of discounted rewards is maximized over time. Alternatively, a way to assign *relative degrees of importance* to rewards that are given at different time steps. (see below)
- **Discounted Return**

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \text{where } 0 \leq \gamma \leq 1 \end{aligned}$$

- **Discount Rate** - real number γ between 0 and 1 inclusive that determines the current value of future rewards
  - γ^{k-1} scales down the reward that would be received at time step k
  - If $\gamma < 1$, then summation has a finite value if the sequence is bounded
  - If $\gamma = 0$, then the agent is greedy and does not care about the future
  - As $\gamma \to 1$, the agent gains more foresight; takes the future into account more strongly

3.4 Unified Notation for Episodic and Continuing Tasks

- To systematically discuss episodic tasks, we consider each episode as a sequence of finite time steps
- By convention, the first time step in each episode is numbered as 0
  - $S_{t,i}$ - state representation at time t, episode i
  - $A_{t,i}$ - action at time t, episode i
  - $R_{t,i}$ - reward at time t, episode i
  - $\pi_{t,i}$ - policy at time t, episode i
  - T_i - final time step of episode i
- To unify notation for returns under episodic and continuing tasks, we define episode termination as entering a special **absorbing state** that transitions only to itself and gives reward of zero.
- Then we define episodic discounted returns using the equation above.

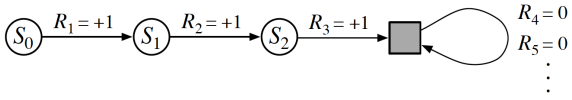


Figure 3: Transition-state diagram

3.5 Policies and Value Functions

- **Policy** - A mapping between states and probabilities of choosing possible actions
  - At time t, $\pi(a|s)$ = probability $A_t = a$ given $S_t = s$
  - If $\pi(a|s) = 1$ for some action at each state, the policy is *deterministic*.
- **Value function** - A function that estimates the "goodness" or value of each state or state-action pair using expected returns. We usually consider value functions assuming a specific agent policy:
  - *State-value function for policy* π - Measures the expected return for each state

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

  - *Action-value function for policy* π - Measures the expected return for each state-action pair

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \text{ for all } (s,a) \in \mathcal{S} \times \mathcal{A} \end{aligned}$$

  - Can be estimated from experience, e.g. Monte Carlo methods to average returns over many episodes
  - Can also represent v_π and q_π as parameterized functions using approximation method (e.g. neural nets)
- **Bellman Update Equations** - *Recursively*-defined value functions
  - **Self-consistency Condition** - A function capable of being defined recursively
  - Form the heart of reinforcement learning updates
  - All value functions must satisfy the self-consistency condition; you can write the function in terms of itself.
  - *Bellman update equation* for $v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} G_t \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} R_t + \gamma G_{t+1} \middle| S_t = s\right] \\ &= \sum_{a,r,s'} p(s', r, a|s)\Big[r + \gamma\, \mathbb{E}[G_{t+1}|S_{t+1} = s']\Big] \\ &= \sum_{a} \pi(a|s) \sum_{r,s'} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned}$$

    Note that we take a sum over a triple (a, r, s') in the third line. This is the same as writing three sums over each variable separately. In other words, the Bellman update takes into account every possible combination of actions, next states, and rewards.
- **Backup Diagrams** - Diagrams that illustrate how future states *backup* or transfer value information to previous states.
  - Can be used to represent Bellman update equations

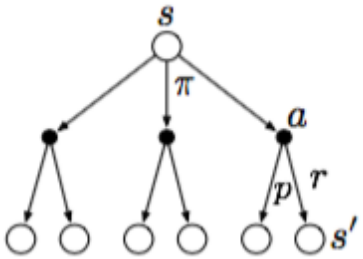


Figure 4: Backup diagram for v_π

3.6 Optimal Policies and Optimal Value Functions

- How do we find good policies using value functions?
- **Definition**: Policy π is better than π' if its expected return is greater than that of π' in *all* possible states. Formally
$$\forall s \in \mathcal{S}\ v_\pi(s) \geq v_{\pi'}(s)$$
- Policy is optimal if there is no policy better than it. There may be multiple, but at least one exists.
- Notation
  - π_* - Space of all optimal policies
  - $v_*(s) \doteq \max_\pi v_\pi(s)$ - *Unique* optimal state-value function (shared by all policies)
  - $q_*(s,a) \doteq \max_\pi q_\pi(s,a)$ - *Unique* optimal action-value function (also shared)
- q_* represents the expected return from taking action a in state s and thereafter following the optimal policy. Thus it can be represented in terms of v_*:
$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(s) | S_t = s, A_t = a] \tag{1}$$
- **Bellman optimality equations** - similar to Bellman update equations, except recursive w.r.t. optimal policy and value function.
  - Can be written without mentioning a specific policy
  - Bellman optimality for state-value function
$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(s) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a) \Big[r + \gamma v_*(s') \Big] \end{aligned}$$
  - Bellman optimality for action-value function
$$\begin{aligned} q_*(s,a) &= \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q(S_{t+1}, a')] \end{aligned}$$
- You can treat the optimality equations like a set of equations. There are $|\mathcal{S}|$ equations (one for each state) and $|\mathcal{S}|$ unknown $v_*(s)$ variables, so given full knowledge of MDP dynamics one can fully solve for $v_*(s)$. The same can be done for $q_*(s,a)$.

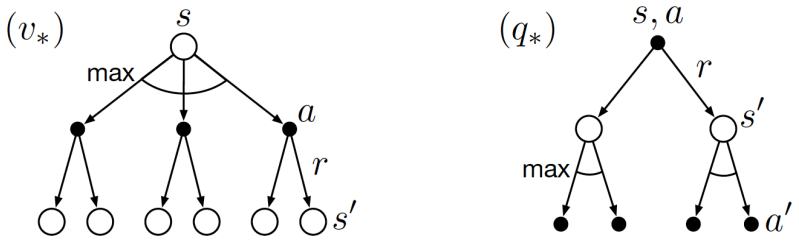


Figure 5: Backup diagrams for v_* and q_*

- π_* should be greedy in terms of v_*, since that gives it the most value, so given v_* or q_* you also know the optimal policy.
  - Given v_* and current state s, do a *one step search*. Choose action a that maximizes expected return.
  - Given q_* and current state s, *you don't even have to search*. The results are 'cached', so you can just pick the $q(s, a)$ values that are maximal for state s.
- We make three large **assumptions** for this solution to work
  (1) Full knowledge of environment dynamics
  (2) Infinite/large computational resources
  (3) **Markov property** - When the *conditional probability distribution* of future states of collection of random variables, **random process**, only depends on the *current state* and nothing before it
    - A mathematical object that has this property is called a **Markov process**
- Many times at least one of these assumptions is violated, so we have to settle for *approximate solutions*
  - e.g. Backgammon has 10^{20} states, which would take thousands of years to solve for v_* using Bellman equations

3.7 Optimality and Approximation

- For large state spaces, computational and memory constraints prevent us from finding completely optimal solutions.
  - Solving for Bellman optimality equations is like an exhaustive search of expected returns; it doesn't scale well computationally.
  - **Tabular methods** represent the state space with a table; this doesn't scale very well memory-wise
- The advantage of approximate solutions is that many states might occur with little or no probability. Since RL learns online, it can learn to focus more on frequently seen states and less on rare ones.
- This distinguishes approximate RL from other ways of solving MDPs.

3.8 Summary

- Reinforcement learning is formally framed using Markov Decision Processes, which describe how agents interact with the environment in order to learn behaviors that maximize reward and achieve their goals.
- Value functions are used to describe the goodness of taking different actions and can be used by the agent to learn optimal policies.
- Optimal solutions are expensive in time and space, but more accurate than approximations. Learning must balance computation costs with degree of optimality as needed.

References

Sutton, Richard S., and Andrew G. Barto. "Finite Markov Decision Processes." Reinforcement Learning: An Introduction, The MIT Press, 2018, pp. 37–57.