Lecture 2 - Markov Decision Process.

1. The Agent-Environment Interface

The learner is called the **agent**. The thing it interacts with is called the **environment**. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards which the agent tries to maximize over time.

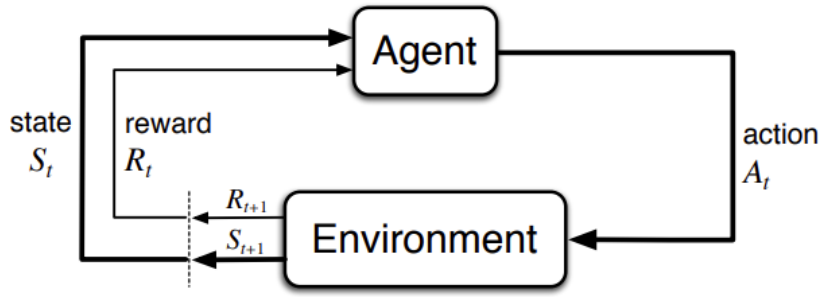

Figure 3.1: The agent–environment interaction in reinforcement learning.

The agent and environment interact at each of a sequence of discrete time steps $t = 0, 1, 2, 3, \ldots$ At each time step t, the agent receives a representation of the **environment's state** $S_t \in S$, and selects an **action** $A_t \in A(S_t)$, where $A(S_t)$ is the set of all actions available in state S_t. One time step later, the agent receives a numerical **reward** $R_{t+1} \in R \subset \Re$, and finds itself in a new state S_{t+1}. A completely specified environment is called a **task, or** an instance of a Reinforcement Learning problem.

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 denoted $\pi_t(a|s)$, that is the probability of taking action $A_t = a$ if $S_t = s$.

Reinforcement learning methods specify how the agent changes its policy over time as a result of experience in order to maximize the total amount of reward it receives over the long run.

2. Goals and Rewards

One distinctive feature of Reinforcement Learning is the use of a reward signal passed from the environment to the agent in order to formalize the idea of a goal. We can state this

informally by what we call the **Reward Hypothesis:**

Every goal can be thought of as maximizing the expected value of the cumulative sum of a received reward.

Even though this may seem limiting, there are plently of examples that show how flexible it really is. For example, to make a robot learn to walk, we can provide a reward on each time step proportional to the robot's forward motion. In making a robot escape from a maze, the reward is often -1 for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible.

If we want to achieve a goal, we must set rewards such that maximizing it implies achieving our goals. The reward signal is **not** the place to impart the agent prior knowledge about **how** to achieve what we want it to do, **instead what** we want to achieve. For example, if we want to train a chess-playing agent, we should reward it only after winning the game, not for achieving subgoals such as taking its opponent pieces or gaining control of the center. This may make the agent biased, and might find ways to maximize the reward even if it costs loosing the game.

3. Returns

The **return** G_t is some function of the reward sequence after time step t denoted by R_{t+1}, R_{t+2}, ...

The **horizon** of a MDP is the number of time steps in each episode (realization of the process). The horizon can be finite or infinite. If it is finite, then the environment is **episodic.** In this case, we have a set of terminal states S', that when reached, a new initial state is sampled from some distribution.

In the case of a **Episodic MDP**, we can define the **Return** as:

$$G_t = \sum_{i=t}^{H-1} R_{i+1}.$$

However in many cases the **horizon** may be infinite. To unify both cases, we can think that whenever we reach the terminal state, we will keep on looping forever in that state and receive 0 reward, thus having a infinite MDP.

Usually, $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$, where γ is a parameter, $0 \leq \gamma \leq 1$ called the *discount rate.*

The discount rate determines the present value of future rewards. If $\gamma = 0$, the agent only cares about immediate reward R_{t+1}. As $\gamma \nrightarrow 1$, the return takes future rewards into account more strongly.

Roughly speaking, the reinforcement learning problem is to choose $A_t,\ A_{t+1},\ ...$ in order to maximize $\mathbb{E}[G_t]$.

4. The Markov Property

A state signal S_t that retains all relevant information about the past is said to be **Markov or to**
have the Markov Property. For example, a chess board position would serve as a Markov State because it summarizes everything important about the complete sequence of positions that led to it. In this case, we say the environment is Markov.

Formally, the environment is Markov if and only if its states satisfies the Markov Property:

$$\Pr(S_{t+1} = s',\ R_{t+1} = r|\ S_0, A_0, R_0, ..., S_t, A_t, R_t\ )\ =\ \Pr(S_{t+1} = s',\ R_{t+1} = r|\ S_t, A_t)\ .$$

5. Markov Decision Processes (MDP)

This is our main mathematical tool to formalize our notion of **environment.** An MDP is a tuple $< \mathcal{S}, A, \mathcal{P}, \mathfrak{R}, \gamma >$.

- S is a finite set of states satisfying the Markov Property.
- A is a finite set of actions
- State transition probabilities $P(s'|s, a)$
- R is a reward function. $r(s, a, s') = \mathbb{E}[R_{t+1}|S_t, A_t, S_{t+1}]$.
- $0 \leq \gamma \leq 1$ is a discount factor.

Note that given the **full Dynamics** of the system $P(S_{t+1},\ R_{t+1}\ |\ S_t, A_t)$, we can compute everything we need such as the state transition probabilities or the reward function. However, we might be given less information. In David Silver course, we are given

- P is a state transition probability distribution $P(s'|s, a)$
- R is a reward function. $r(s, a) = \mathbb{E}[R_{t+1}|S_t, A_t]$.

instead of the full system dynamics.
A **transition graph** is a useful way of summarizing the dynamics of a finite state MDP.

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 3.1: 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, $a \in \mathcal{A}(s)$.

Figure 1: Transition Table for MDP

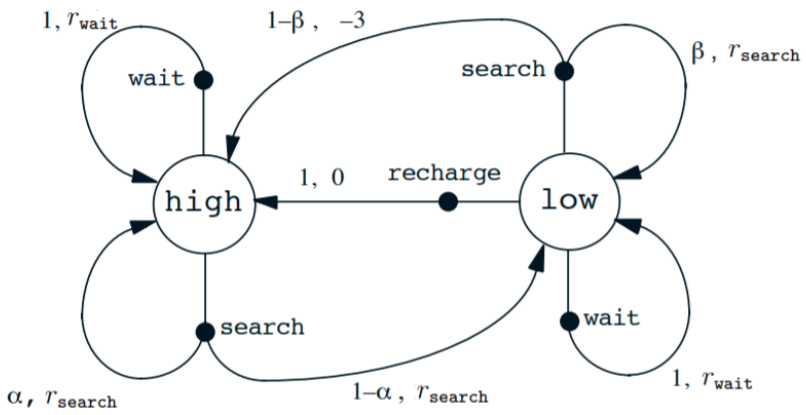

Figure 3.3: Transition graph for the recycling robot example.

Figure 2: Transition Graph for MDP

6. Value Functions

The **value** of a state s under a policy π, denoted by $v_\pi(s)$ is the **expected return when starting in state s and following policy π.**

Formally, $v_\pi(s) = \mathbb{E}_\pi(G_t|S_t = s)$.

Similarly, we define the value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, as t**he expected return starting from s, taking the action a, and thereafter following policy π.**

Formally, $q_\pi(s,a) = \mathbb{E}_\pi(G_t|S_t = s,\ A_t = a)$.

The value function can be recursively decomposed into two parts:

- Immediate Reward R_{t+1}
- Discounted value of Successor state $\gamma\ v(S_{t+1})$

$$v_\pi(s) = \mathbb{E}_\pi(G_t|S_t = s) = \mathbb{E}[R_{t+1} + \gamma\ \ v(S_{t+1}) \mid S_t = s].$$

Or equivalently,

$$v_\pi(s) = \sum_{a\in A} \pi(a|s)\, q_\pi(s,a) = \sum_{a\in A} \pi(a|s) \left[r(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) v_\pi(s') \right].$$

This is the **Bellman Expectation Equation** for $v_\pi(s)$. We can use them to verify if a given function is a value function for the MDP.

One way to visualize these equations is through **Backup Diagrams** or Backup Operations.

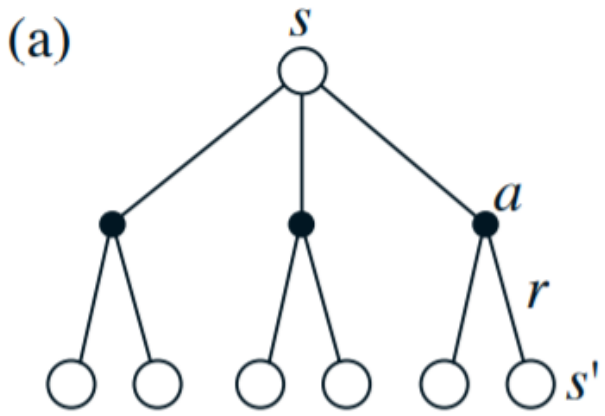

Figure 3: Backup Diagram for $v_\pi(s)$

The Bellman equation for v_π expresses a relationship between the value of a state and the values of its successor states. We can think of looking ahead from one state to its possible successor states, as suggested by the Backup Diagram.

Each open circle represents a state and each solid circle represents a state–action pair. Starting from state s, the root node at the top, the agent could take any of some set of actions. From
each of these, the environment could respond with one of several next states s', along with a reward, r. The Bellman equation averages over all the possibilities, weighting each by its probability of occurring.

In a similar way, we have a Bellman Expectation Equation for the Action-Value function along with its Backup Diagram.

$$q_\pi(s,a) = \mathbb{E}_\pi(G_t|S_t = s,\ A_t = a) = \mathbb{E}_\pi(R_{t+1} + \gamma\, q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a)$$

or Equivalently,

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') q(s',a')$$

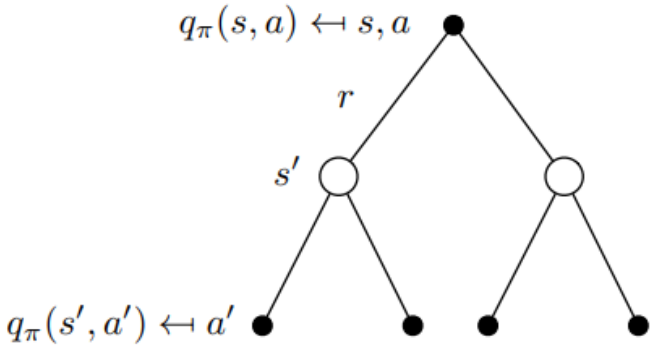

Figure 4: Backup Diagram for Action-Value Function

7. Optimal Policies and Value Functions

The **Optimal State-Value Function** $v_*(s) = \max_\pi v_\pi(s)$ is the maximum value we can extract from the MDP at every state.

The **Optimal Action-Value Function** denoted $q_*(s,a) = max_\pi q(s,a)$ represents the

maximum value we can extract from the MDP at every state if we commit to a certain action.

We can define a **partial ordering** over policies where $\pi \geqslant \pi' \; if \; v_\pi(s) \geqslant v_{\pi'}(s), \; \forall s \in S.$

There is a **Theorem** that states that for any MDP:

- There **exists** an optimal policy π_* such that $\pi_* \geqslant \pi, \; \forall \pi.$
- All optimal policies **achieve** the optimal state-value function $v_{\pi*}(s) = v_*(s)$ (Iff)
- All optimal policies **achieve** the optimal action-value function $q_{\pi*}(s,a) = q_*(s,a)$ (Iff)

This theorem implies that the Optimal State-Value and Optimal Action-Value functions correspond to values obtained if we follow the optimal policy, and that if we find a policy that ahieves those values, then it is optimal.

There is always a **deterministic optimal policy** for an MDP, and can **be found by maximising over** $q_*(s,a)$ in the following way:

$$\pi_*(a|s) = 1 \; if \; a = argmax_a \; q_*(s,a).$$

This means that if we find $q_*(s,a)$ we have solved the MDP.

8. Relationships between Optimal Value functions

The following recursive relationship holds:

$$v_*(s) = max_a \; q_*(s,a).$$
$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_*(s').$$

By comining these 2 equations, we obtain recursive relationships for v_* and q_*.

$$v_*(s) = \mathbb{E}_{\pi*}[R_{t+1} + \gamma \, v_*(S_{t+1}) | S_t = S, A_t = a] = max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_*(s') \right].$$

Bellman Optimality Equation for v_*

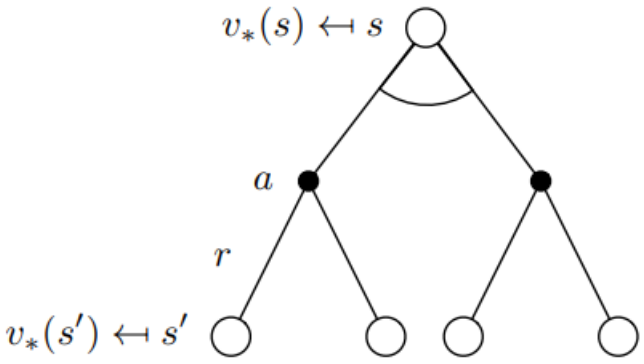

Figure 5: Backup Diagram for Bellman Optimaly Equation for State-Value Function

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \, max_{a'} \, q_*(S_{t+1}, a')] = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \, max_{a'} \, q_*(s',a') \, .$$

Bellman Optimality Equation for q_*

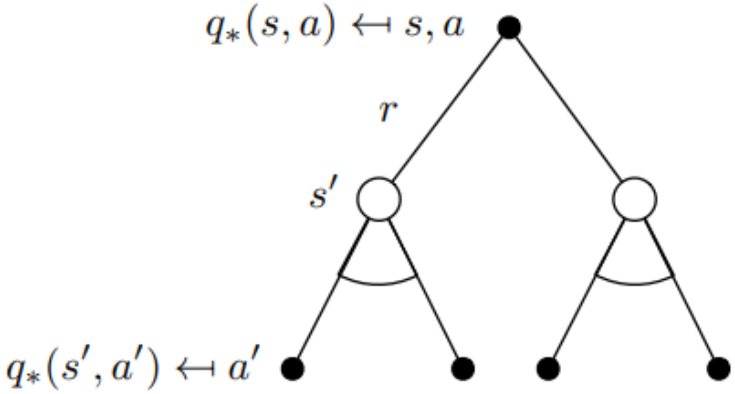

Figure 6: Backup Diagram for Bellman Optimaly Equation for Action-Value Function

These equations are **non-linear**. There is no closed solution. Instead, we will use iterative methods. Then, because of the theorem stated, if we find q_* we will find the optimal policy.