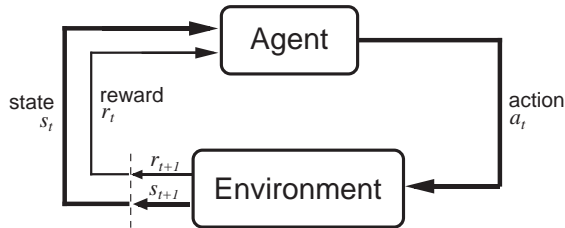


Reinforcement Learning



r_t	numerical reward received at time step t
$s_t \in S$	state at time t
$a_t \in A$	action taken at time t
R_t	total reward received after time t
$\pi(s,a)$	<i>policy</i> : probability of taking action a in state s
$V^\pi(s)$	<i>value of state s</i> : expected return, starting at state s and following policy π
$Q^\pi(s,a)$	<i>value of (s,a) pair</i> : expected return, starting at state s , taking action a , then following policy π

Markov Property

- History of states, actions and rewards for a decision process:

$$\langle s_0, a_0, 0 \rangle \langle s_1, a_1, r_1 \rangle \langle s_2, a_2, r_2 \rangle \dots \langle s_t, a_t, r_t \rangle \langle s_{t+1}, a_{t+1}, r_{t+1} \rangle$$
- This process satisfies the **Markov Property** if the probability of a particular state s' and reward r' at time $t+1$ depends only on the state and action at time t
- If a reinforcement learning task satisfies the Markov Property, it is a **Markov Decision Process** (MDP)
- If the states and actions are finite, the process is a **finite MDP** and is completely specified by S , A , and the following “one-step” transition probabilities and expected rewards:

$$\Pr \{s_{t+1} = s' \mid s_t = s, a_t = a\} \text{ for all } s, s' \in S, a \in A(s)$$

$$E \{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \text{ for all } s, s' \in S, a \in A(s)$$

Finite MDP Example

Recycling Robot



At each step, robot has to decide whether to:

- (1) actively search for a can
- (2) wait for someone to bring a can
- (3) go to home base and recharge

- Searching is better but runs down the battery – if robot runs out of power while searching, it has to be rescued (bad!)
- Decisions are based on current energy level: **high**, **low**
- Reward = number of cans collected

Recycling Robot MDP

$S = \{\text{high}, \text{low}\}$

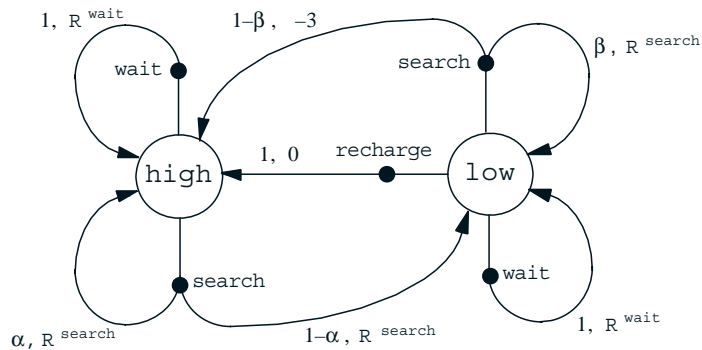
$A(\text{high}) = \{\text{search}, \text{wait}\}$

$A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$

$R^{\text{search}} =$ expected no. of cans while searching

$R^{\text{wait}} =$ expected no. of cans while waiting

$R^{\text{search}} > R^{\text{wait}}$



Goals of Reinforcement Learning

Recall the following important functions:

$\pi(s,a)$ *policy*: probability of taking action a in state s

$V^\pi(s)$ *value of state s* : expected return, starting at state s and following policy π

$Q^\pi(s,a)$ *value of (s,a) pair*: expected return, starting at state s , taking action a , then following policy π

Our goal is to compute the ***optimal policy*** π^* that has the ***optimal state-value and action-value functions***:

$$V^*(s) = \max_{\pi} V^\pi(s) \quad \text{for all } s \in S$$

$$Q^*(s,a) = \max_{\pi} Q^\pi(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

Monte Carlo Methods



- Named after the city in Monaco
- 1940's: von Neumann developed formal foundations
- Used to simulate the outcome of probabilistic processes

- Begin with ***random policy***:
for each state, every action is equally probable
- Episodic process: randomly generate ***many*** episodes with expected rewards:
one episode: $\langle s_0, a_0, 0 \rangle \langle s_1, a_1, r_1 \rangle \dots \langle s_T, a_T, r_T \rangle$
- For each state-action pair (s,a) , record all observed returns*
- After many episodes, determine expected return for each (s,a) pair, and ***adjust policy to increase likelihood of (s,a) pairs with large returns***
- Repeat the process with improved policy

Q-Learning Method

Principles:

- If an action in a given state causes something bad to happen, learn not to do that action in that situation
- If an action in a given state causes something good to happen, learn to do that action in that situation
- If all actions in a given state cause something bad to happen, learn to avoid that state (i.e., don't take actions in other states that would lead you to be in that bad state)
- If any action in a given state causes something good to happen, learn to like that state

Value functions are recursive...

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \dots) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^\pi(s) &= E_\pi \{ R_t \mid s_t = s \} \\ &= E_\pi \{ r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s \} \end{aligned}$$

...

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \quad \text{for all } a_{t+1} \in A$$

Learning Algorithm for $Q^*(s,a)$

Initialize table of $Q(s,a)$ values to 0

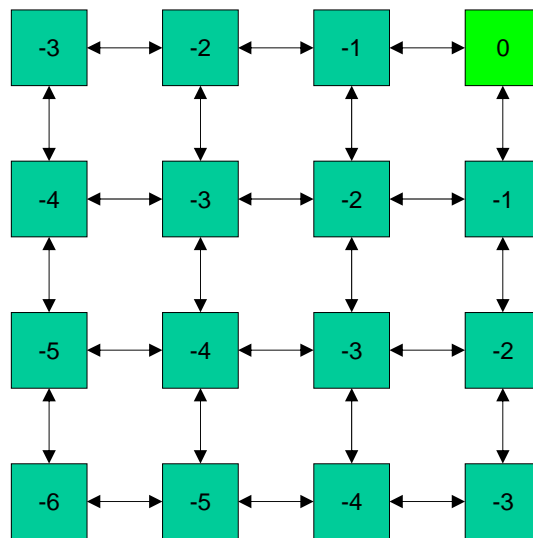
1. From current state s , select an action a and execute it
2. Observe the immediate reward r
3. Observe the new state s'
4. Update the table entry for $Q(s,a)$ using the relation:

$$Q'(s,a) = r + \gamma \max_{a'} Q(s',a') \quad \text{for all } a' \in A$$

5. Update the state to s'
6. Loop back to step 1 and repeat until $Q(s,a)$ stabilizes

If we have the optimal Q function, $Q^*(s,a)$, how can we derive the optimal policy π^* ?

Value functions



The *value function* represents how good a state is in the long run. It reflects the overall reward that can be accumulated moving from the given state to the goal

The numbers in the states represent the *optimal value function* to reach the goal from each state

The value function for each state is the shortest number of steps to the goal, multiplied by -1