Rewards

Reward hypothesis: "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)"

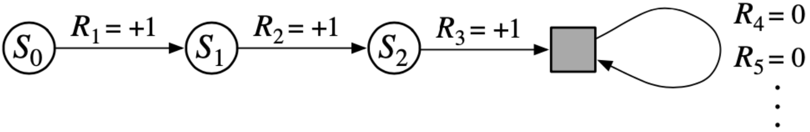
So in an MDP, the reward should represent the *goal* we want the agent to accomplish

- Not e.g. some information about *how* it should reach the goal

Episodic tasks

Suppose the interaction ends after a finite number of time steps (e.g. in tasks where a goal is eventually reached)

- Each such subsequence of time steps is called an episode
- Termination is marked by transitioning to a special state: the terminal state; all transitions from this state go back to this state, and give reward 0
- State set notation: $\mathcal{S}^+ := \mathcal{S} \cup \{\text{terminal state}\}$



After an episode is terminated, we can think of the agent as starting a new episode (where it can apply what it learned in previous episodes)

Returns in episodic tasks

Return: rewards from time t to the end

$$G_t := R_{t+1} + R_{t+2} + \ldots + R_T$$

Here T is the termination time (a random variable): the time step where the terminal state is first reached

So:

- reward: in one time step
- return: until the end
- value: expected return

Continuing tasks

If a task might continue infinitely long, it is called a continuing task

Issue: sum of rewards might be infinite/undefined!

Solution: use discounting with some chosen rate $0 \leq \gamma \leq 1$, and define G_t as

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

Expresses that rewards farther in the future should be given less weight

Special cases:

- $\gamma = 1$: no discounting, ordinary sum
- $\gamma = 0$: care only about immediate future (take $0^0 = 1$; called "myopic")

Policies

A policy specifies how an agent chooses actions

- May be probabilistic in general (like the ϵ-greedy strategies we saw before)
- Notation: $\pi(a \mid s) := P(A_t = a \mid S_t = s)$ (for all t)

If we know p and π, we can compute probabilities of all future events given a current state
Exercise 3.11 (section 3.5): If the current state is S_t, and actions are selected according to stochastic policy π, then what is the expectation of R_{t+1} in terms of π and the four-argument function p (3.2)? ($p(s', r \mid s, a)$)

Value functions

For known p and π, the value (expected return) is described for each current state by

$$v_\pi(s) := E_\pi[G_t \mid S_t = s]$$

(the state-value function for policy π)

Similarly, if both S_t and A_t are known we have the action-value function for policy π:

$$q_\pi(s, a) := E_\pi[G_t \mid S_t = s, A_t = a]$$

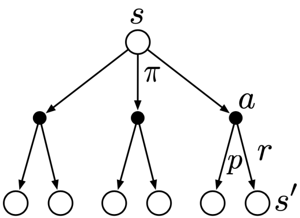
Bellman equations

Rewriting $v_\pi(s)$, we find

$$\begin{aligned} v_\pi(s) &:= E_\pi[G_t \mid S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned}$$

Called the Bellman equation for v_π

Plays a central role in many reinforcement learning methods



backup diagram for v_π

Optimal policies and value functions

A policy π is at least as good as another policy π' if

$$v_\pi(s) \geq v_{\pi'}(s) \qquad \text{for all } s$$

A policy that is at least as good as *all* other policies is an optimal policy (written π_*)

- one always exists!
- there may be more than one

All optimal policies have the same optimal (state/action-)value functions:

$$v_*(s) := \max_\pi v_p i(s) \qquad q_*(s, a) := \max_\pi q_\pi(s, a)$$

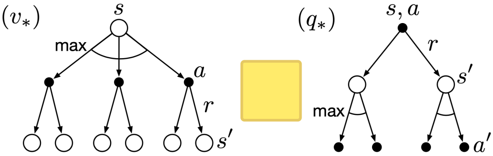
Bellman optimality equation

Bellman equation for v_*:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\
&= \max_a E_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a E_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a E[R_{t+1} + \gamma v_*(S_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r \mid s, a) \, [r + \gamma v_*(s')]
\end{aligned}
$$

(and similar for q_*)

In backup diagrams, arcs represent taking maximum:



backup diagrams for v_* and q_*

Computing optimal policies and value functions

- Using the Bellman equations, it is in principle possible to compute v_* and q_*
  - Assuming p is known
  - May take a prohibitive amount of computation time and memory if number of states / actions is large
- Once we know v_*, we can compute an optimal policy: for each state, compare each action's expected immediate reward plus the expected return after that (i.e. greedy w.r.t. v_*)
- If we know q_*, this becomes even easier

In reinforcement learning, p is rarely known in practice. Approach: estimate it from experience, and use it to in turn approximate $v_*/q_*/\pi_*$

Prediction

prediction: problem of finding an estimate V of v_π for a given policy π

Possible approaches:

- Dynamic programming (chapter 4; assumes p is known)
  - Keep track of our estimates V so far
  - keep improving (for all states s) our estimates $V(s)$ by using new estimates $V(s')$ for successor states
- Monte Carlo methods (chapter 5)
  - Play through an episode until termination
  - for each state s visited during the episode, we know the return we got, and can use this to improve our estimate $V(s)$

Prediction in temporal-difference learning

Temporal-difference (TD) learning: combines ideas from dynamic programming (DP) and Monte Carlo (MC) methods:

- Like MC, based on experience rather than knowledge of p
- Unlike MC, don't wait for an episode to end before we update V

One-step TD

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

estimate on return from now on until the end

Notes:

- $V(S)$ is updated using a version of the incremental formula from chapter 2
- Like DP, TD *bootstraps*: uses previous estimates for other states to form a new estimate (MC doesn't do this)

Control: generalized policy iteration

Next to *prediction*, there is the control problem: approximating optimal policies

Common idea for DP/MC/TD: generalized policy iteration (GPI)

- keep track of a policy π and estimated (action) values Q
- repeatedly improve the estimate Q for the current policy π (the prediction problem)
- at the same time, repeatedly improve π by making it closer to the greedy strategy w.r.t. Q

TD control: Sarsa

Estimates Q instead of V, using a similar formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

Named Sarsa because it uses $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

Sarsa algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

On- vs. off-policy control

Sarsa estimates Q for the policy it's currently following: example of an on-policy method

Sometimes we want to estimate values for a different policy than the one we're following

- Common reason: the policy we're following needs to do exploration, but we know this makes it worse at exploitation

Q-learning

Q-learning: estimate the optimal policy's q_*, regardless of the policy being followed

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Expected Sarsa

Expected Sarsa: estimate any target policy's q_π, regardless of the policy being followed

Update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma E_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\right]$$

More general than Q-learning