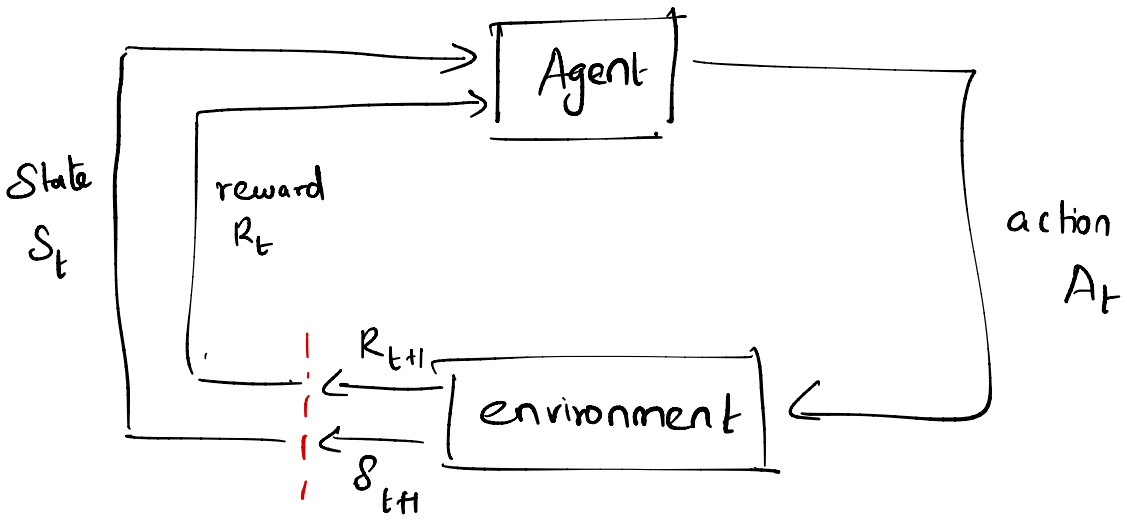
3. Finite Markov Decision Processes

Markov Decision Process (MDP): A mathematical formalization of sequential decision making / RL.



The agent and the 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 some representation of the environment's state $S_t \in \mathcal{S}$.

→ on that basis, selects an action $A_t \in \mathcal{A}(s)$

→ in next time step, receives a numerical

reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and finds itself in a new state S_{t+1}.

Sequence / trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots$

Finite MDP: $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ — all have finite number of elements.

R_t, S_t — well defined discrete probability distributions dependent only on the preceding state and action.

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

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

$p \Rightarrow$ defines the dynamics of the MDP.

$p: \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is a deterministic fn. of 4 arguments.

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

From the dynamics fn 'p' one can compute anything else one might want to know about the environment.

State-transition probability: $p : S \times S \times A \to [0, 1]$

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

$$= \sum_{r \in R} p(s', r | s, a)$$

Expected reward for s, a pair: $r : S \times A \to \mathbb{R}$

$$r(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$$

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

Expected reward for state-action-next state triple:

$$r : S \times A \times S \to \mathbb{R}.$$

$$r(s, a, s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']$$

$$= \sum_{r \in R} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

Markov property :-

$$P(S_t, R_t \mid S_{t-1}, A_{t-1}) = P(S_t, R_t \mid S_{t-1}, A_{t-1}, R_{t-1}, S_{t-2}, A_{t-2}, \ldots)$$

S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}.

Note: This is a restriction on the state not on the decision process.

The MDP framework is abstract and flexible.

Time steps → need not refer to fixed interval of time.

actions → can be low-level control or high-level decisions.

states → low level sensations or high level object representations.

Also, the agent/environment boundary is not physical.

Anything - that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of the environment.

The boundary represents the limit of the agent's absolute control, not of its knowledge.

For example, the agent often knows about how rewards are computed. Still rewards are external since the agent cannot change it arbitrarily. Sometimes agent might know "everything" about how its env. works and still face a difficult RL task. ex: You know how rubik's cube work still do not know how to solve it.

MDP: reduces any problem of learning goal directed behaviour to three signals:

- one signal to represent the choices made by the agent (the actions)
- one signal to represent the basis on which the choices are made (the states)
- one signal to define the agent's goal (the rewards)

How to represent states and action? We will assume the representation is given so that we can focus on the general principles for learning how to behave once the representations have been selected. How to learn these representation is a topic for future discussions.

Goals and rewards:-

Reward is your way of communicating

to robot what you want it to achieve, not how you want it achieved.

How ⇒ Prior knowledge. Can be inserted to initial policy or initial value function.

Returns and Episodes :-

Goal of the agent : To maximize the cumulative reward it receives in the long run.

Cumulative reward = return.

(1)

$$\text{Expected return } G_t = R_{t+1} + R_{t+2} + \dots + R_T$$

where T is the final step. This definition assumes that the interaction ends at some point T.

ex: Plays of the game.

Such repeated but finite-time interactions are called "episodes".

Each episode ends in a terminal state, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states.

S - set of all non-terminal states

S^+ - set of all states + terminal state.

T is a random variable that varies episode to episode.

These tasks are called episodic tasks.

What if a task goes on continually without limit?

ex: An application to a robot with a long life span.

These are called continuing tasks.

Return eqn. (1) is problematic for continuing tasks since $T = \infty$.

Idea of discounting :-

discounted return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \text{---(2)}$$

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

γ determines the present value of future rewards.

Reward received after k timesteps is worth only γ^{k-1} times its worth.

If $\gamma < 1$, the infinite sum (2) has a finite value as long as the reward sequence $\{R_k\}$ is bounded.

$\gamma = 0 \Rightarrow$ the agent is "myopic". maximizes only R_{t+1}.

$\gamma \to 1 \Rightarrow$ the agent becomes more farsighted.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

$$= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right)$$

$$\boxed{G_t = R_{t+1} + \gamma G_{t+1}}$$

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

ex: if reward is constant $+1$, then

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

Unified notation for episodic and continuing tasks:-

episodic tasks: $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$

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

Consider episode termination to be entering of a special absorbing state that transitions only to itself and that generates only rewards of zero.

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

including the possibility that $T = \infty$ or $\gamma = 1$ (but not both). Can we have both $T = \infty$ and $\gamma = 1$? More on this later.

Policies and Value functions :-

Almost all RL algorithms involve estimating Value functions - functions of states (or state-action pairs) that estimate how good it is for the agent to be in a given state (or

how good it is to perform a given action in a given state).

Value fns are defined w.r.t. policies.

Policy: mapping from states to probabilities of selecting each possible action.

$\pi(a|s)$ is probability that $A_t = a$ if $S_t = s$.

State value function :-

$V_\pi(s)$: value fn. of a state s under a policy π.

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \quad \forall\, s \in S$$

Note: V_π (terminal state) $= 0$.

Action-value function :-

$q_\pi(s,a)$ = value of taking action a in state s.

= expected return starting from s, taking the action a, and thereafter following policy π.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a \right]$$

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

V_π and q_π can be estimated from experience.

One can maintain sample averages of actual returns for each state encountered. As $T \to \infty$, these averages will converge to actual values. This is called Monte-Carlo method (more later).

However, this solution is not practical if there are very many states. Then we should rely on parameterized functions for v_π and q_π (again seen later).

For any policy π, and any state s,

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s \right]$$

$$= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right]$$

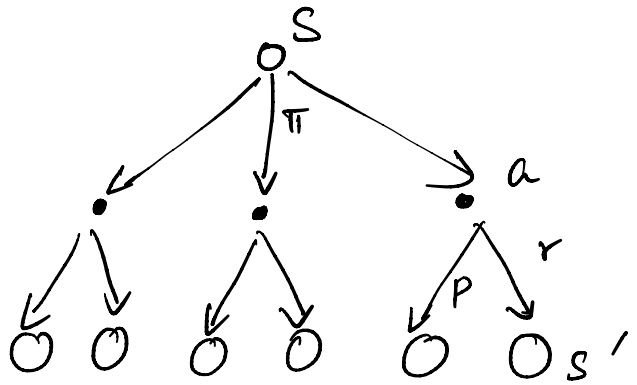
$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s' \right] \right]$$

Bellman equation for v_π.

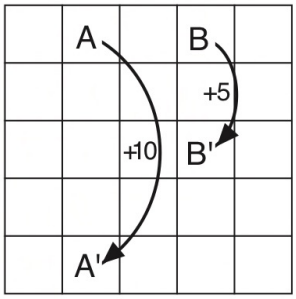
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right] \quad \forall s \in \mathcal{S}.$$

Note: v_π is the unique soln to its Bellman equation.

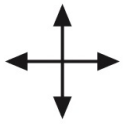
Backup diagram for v_π:



The Bellman equation averages over all the possibilities.



Gridworld



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

state value fn. for an equiprobable random policy.

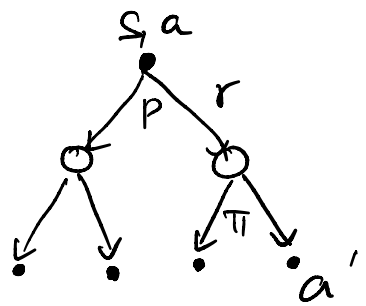
Gridworld: −1 for actions that would take the agent off the grid and leave state unchanged.

0 for all other actions except for states A and B.

Bellman equation for $q_\pi(s,a)$:

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

backup diagram :



Optimal policies and optimal value fns :-

A policy π is defined to be better than or equal to π' if its expected return is greater than or equal to that of π' for all states.

$\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ $\forall s \in S$.

There is always at least one policy that is better than or equal to all other policies.

This is an optimal policy.

π_* – optimal policy.

V_* – State-value fn of the optimal policy.
(or) Optimal state value fn.

$$V_*(s) = \max_{\pi} V_\pi(s) \quad \forall s \in S.$$

Optimal policies also share the same optimal action-value fn. q_*.

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \quad \forall s \in S, \; a \in A.$$

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma V_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a \right]$$

Because v_* is the value fn for a policy, it must satisfy the self-consistency condition given by the Bellman equation.

Because it is the optimal value fn, v_*'s Consistency condn. can be written without reference to any specific policy.

$$v_*(s) = \max_{a \in A} q_{\pi_*}(s,a)$$

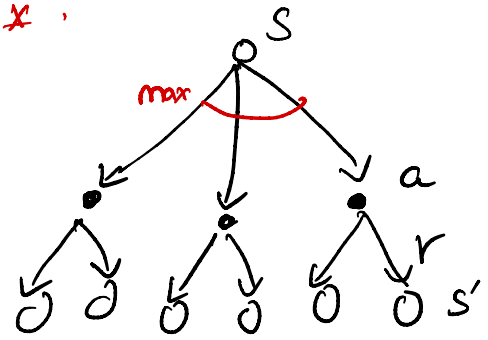
$$= \max_a \mathbb{E}_{\pi_*}\left[G_t \mid S_t = s, A_t = a \right]$$

$$= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a \right]$$

$$v_*(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a \right]$$

$$v_*(s) = \max_a \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma v_*(s') \right]$$

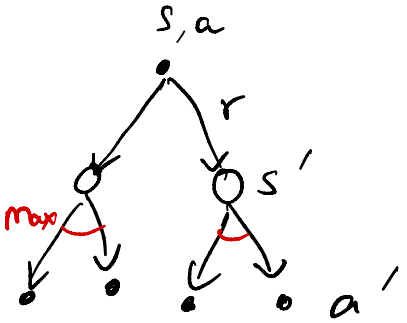
Bellman optimality eqn. for v_*.



$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a \right]$$

$$= \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right]$$

Bellman optimality eqn for q_*.



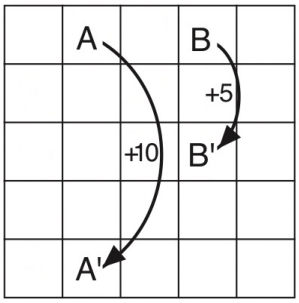
Note: For finite MDPs, Belleman optimality eqn for V_π has a unique soln. independent of the policy.

It is a system of eqns with 'n' equations for 'n' states with 'n' unknowns.

If 'p' is known, one can solve this by treating it as system of non-linear eqns.

Once one has V_*, then the optimal policy can be easily constructed by acting greedily w.r.t. V_*. One-step look ahead is enough to act optimally.

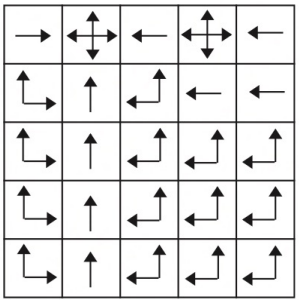
Having q_* makes it even easier. No need for one step look ahead. But the tradeoff is you have to store q value for all actions in every state.



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

Explicitly solving the Bellman optimality eqns. gives us the optimal policy and hence solves the RL problem.

However, it has 3 assumptions which are rarely true in practice.

(1) We accurately know the dynamics of the env.

(2) We have enough Compute to complete the Computation of the soln.

(3) the Markov property.

One of these assumptions would be violated in most of the interesting applications.

Ex: Playing backgammon

(1), (3) are ok. But for (2), there are 10^{20} states!

All of RL is about approximately solving this Bellman optimality eqn. Note that the agent need not act optimally in every possible situation one can think of. It is enough

if it acts optimally in states that it visits frequently. This is the key property we will exploit while developping approx. Solutions.