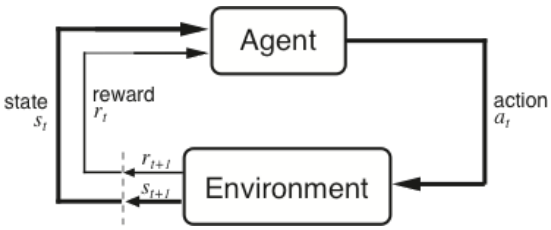
The Reinforcement Learning Problem

Suggested reading:

Chapter 3 in R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction
MIT Press, 1998.

The Reinforcement Learning Problem



Contents:

The Reinforcement Learning Problem

Objectives of this chapter:

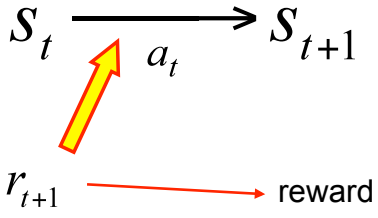
- describe the RL problem we will be studying for the remainder of the course
- present idealized form of the RL problem for which we have precise theoretical results;
- introduce key components of the mathematics: value functions and Bellman equations;
- describe trade-offs between applicability and mathematical tractability.

The Agent-Environment Interface

S set of possible states $s_t \in S$

$A(s_t)$ set of actions available in state at time t $a_t \in A(s_t)$

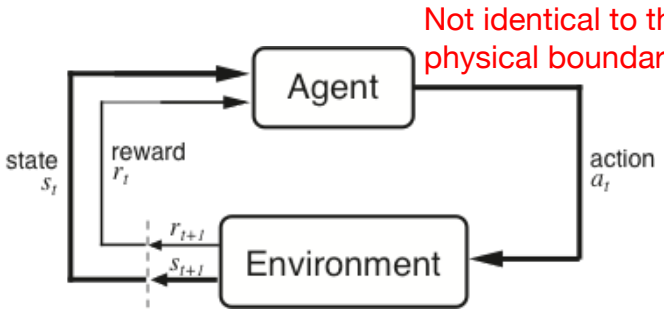
$$A = \bigcup_{s_t \in S} A(s_t)$$



$s_t, s_{t+1} \in S$

$t = 0,1,2,...$

The Agent-Environment Interface



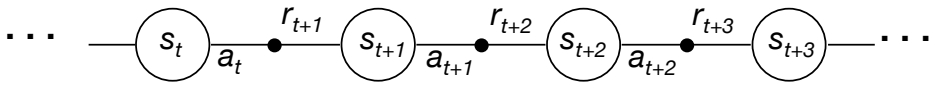
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

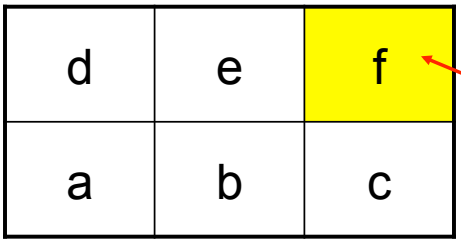
produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}



Example 1



$S = \{a, b, c, d, e, f\}$

terminal state

$A = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$

$A(a) = \{\uparrow, \rightarrow\}$

$A(b) = \{\uparrow, \rightarrow, \leftarrow\}$

$A(c) = \{\uparrow, \leftarrow\}$

$A(d) = \{\downarrow, \rightarrow\}$

$A(e) = \{\downarrow, \rightarrow, \leftarrow\}$

$r = 100 \Rightarrow c \rightarrow f,\ e \rightarrow f$

$r = 0 \Rightarrow$ else

Example 1

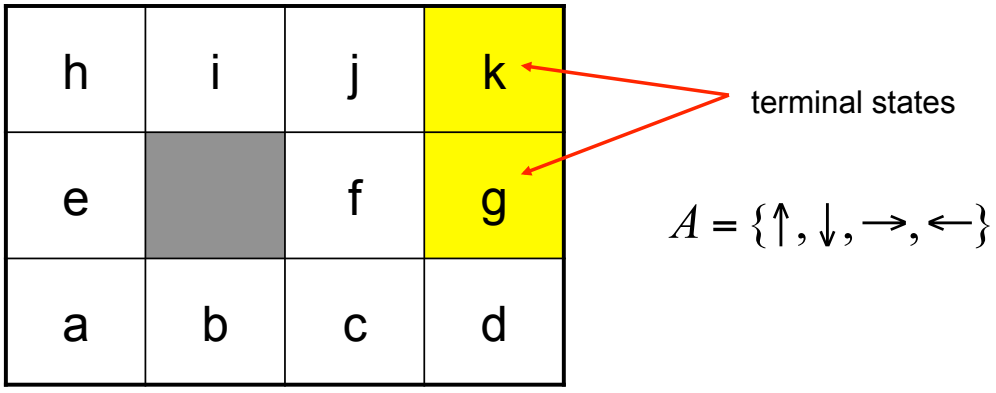
d	e	f
a	b	c

episodes:

$$a \rightarrow d \rightarrow e \rightarrow f$$

$$a \rightarrow b \rightarrow a \rightarrow d \rightarrow e \rightarrow b \rightarrow c \rightarrow f$$

Example 2



$r = 100 \Rightarrow j \rightarrow k$

$r = -100 \Rightarrow f \rightarrow g \quad d \rightarrow g$

$r = 0 \Rightarrow$ else $(r = -20)$

Policy

Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s,a) =$ probability that $a_t = a$ when $s_t = s$

$$s_t \in S \longrightarrow a_t \in A(s)$$

$$\pi^* : S \to A \qquad \text{optimal policy}$$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

Example 1

d	e	f
a	b	c

an optimal policy:

$$\pi_1(a) = \uparrow$$

$\pi_1(a,\uparrow) = 1 \qquad \pi_1(a,\rightarrow) = 0$

$$\pi_1(b) = \rightarrow$$

$$\pi_1(c) = \uparrow$$

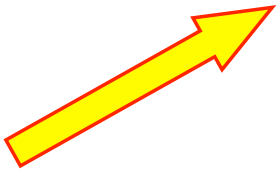
$$\pi_1(d) = \rightarrow$$

$$\pi_1(e) = \rightarrow$$

Example 1

d	e	f
a	b	c

random policy



$$\pi_2(a,\uparrow) = \pi_2(a,\rightarrow) = \frac{1}{2}$$

$$\pi_2(b,\uparrow) = \pi_2(b,\rightarrow) = \pi_2(b,\leftarrow) = \frac{1}{3}$$

$$\pi_2(c,\uparrow) = \pi_2(c,\leftarrow) = \frac{1}{2}$$

$$\pi_2(d,\downarrow) = \pi_2(d,\rightarrow) = \frac{1}{2}$$

$$\pi_2(e,\downarrow) = \pi_2(e,\rightarrow) = \pi_2(e,\leftarrow) = \frac{1}{3}$$

Getting the Degree of Abstraction Right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), “mental” (e.g., shift in focus of attention), etc.
- States can be low-level “sensations”, or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being “surprised” or “lost”).
- An RL agent is not like a *whole* animal or robot.
- Reward computation is in the agent’s environment because the agent cannot change it arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.

Goals and Rewards

- Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.
- A goal should specify **what** we want to achieve, not **how** we want to achieve it.
- A goal must be outside the agent's direct control—thus outside the agent.
- The agent must be able to measure success:
  - explicitly;
  - frequently during its lifespan.

Returns

Suppose the sequence of rewards after step t is:

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

$$R_t = r_{t+1} + r_{t+2} + \ldots$$

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

The present return formulation is problematic for continuing tasks because the final time step would be T=∞ and the return, which is what we are trying to maximize, could itself easily be infinite.

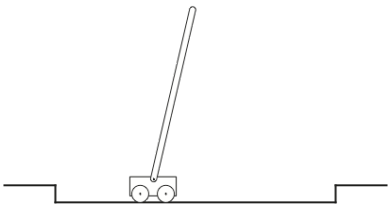
Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma, 0 \le \gamma \le 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

An Example



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

reward $= +1$ for each step before failure

$\Rightarrow$ return = number of steps before failure

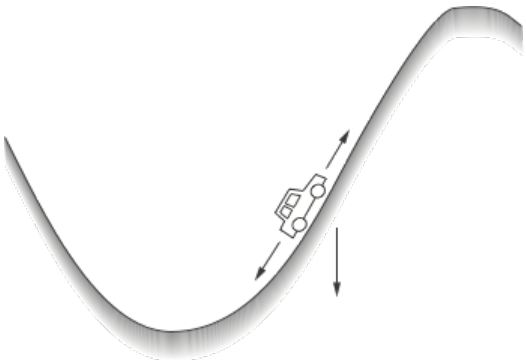
As a **continuing task** with discounted return:

reward $= -1$ upon failure; 0 otherwise

$\Rightarrow$ return $= -\gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill as quickly as possible.

reward $= -1$ for each step where **not** at top of hill

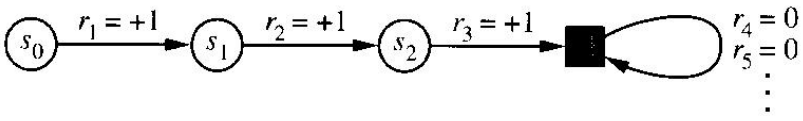
$\Rightarrow$ return $=$ $-$ number of steps before reaching top of hill

Return is maximized by minimizing number of steps to reach the top of the hill.

A Unified Notation

where γ can be 1 only if a zero reward absorbing state is always reached.

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have distinguish between episodes, so we write S_t instead of $S_{t,j}$ for the state at step t of episode j.
- Think of each episode as ending in an absorbing state that always produces reward of zero:



- We can cover all cases by writing

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \qquad T = \infty \text{ xor } \gamma = 1$$

where γ can be 1 only if a zero reward absorbing state is always reached.

The Markov Property

- "The state" at step *t*, refers to whatever information is available to the agent at step *t* about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\Pr\left\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\} = \Pr\left\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\right\}$$

$$\text{for all } s', r, \text{ and histories } s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0.$$

Even when the state signal is non-Markov, it is still appropriate to think of the state in reinforcement learning as an approximation to a Markov state.

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$\mathbf{P}^a_{ss'} = \Pr\left\{s_{t+1}=s' \mid s_t=s, a_t=a\right\} \quad \text{for all } s, s' \in S, a \in A(s).$$

  - **reward probabilities (expectation)**:

$$\mathbf{R}^a_{ss'} = E\left\{r_{t+1} \mid s_t=s, a_t=a, s_{t+1}=s'\right\} \quad \text{for all } s, s' \in S, a \in A(s).$$

deterministic:

$$P^a_{ss'} = \begin{cases} 1 & \text{if } s' = s(a) \\ 0 & \text{else} \end{cases} \qquad R^a_{ss'} = r(s,a)$$

An Example Finite MDP

Recycling Robot:

- At each step, robot has to decide whether it should
  - (1) actively search for a can,
  - (2) wait for someone to bring it a can, or
  - (3) go to home base and recharge.
- Searching is better but runs down the battery; if the robot runs out of power while searching, he has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

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

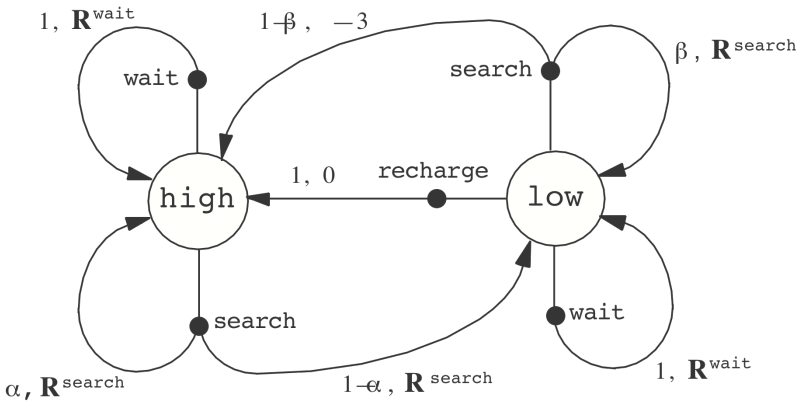
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

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

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

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

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



Recycling Robot MDP

Transition probabilities and expected rewards for the finite MDP of the recycling robot example:

s	s'	a	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\text{search}}$
high	low	search	$1-\alpha$	$\mathcal{R}^{\text{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$\mathcal{R}^{\text{search}}$
high	high	wait	1	$\mathcal{R}^{\text{wait}}$
high	low	wait	0	$\mathcal{R}^{\text{wait}}$
low	high	wait	0	$\mathcal{R}^{\text{wait}}$
low	low	wait	1	$\mathcal{R}^{\text{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0

Value Functions V and Q

The **value of a state** is the expected return starting from that state. It depends on the agent's policy:

State - value function for policy π :

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\}$$

The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π :

Action - value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Value Functions

The value functions can be estimated from experience.

- If an agent follows policy and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $V^{\pi}(s)$, as the number of times that state is encountered approaches infinity.
- If separate averages are kept for each action taken in a state, then these averages will similarly converge to the action values, $Q^{\pi}(s,a)$.

→ Monte Carlo methods because they involve averaging over many random samples of actual returns

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships.

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \cdot Q^{\pi}(s,a)$$

?

Example 1- Value Function V

π - random

$r = 100$ → $c \to f$, $e \to f$

$r = 0$ → else

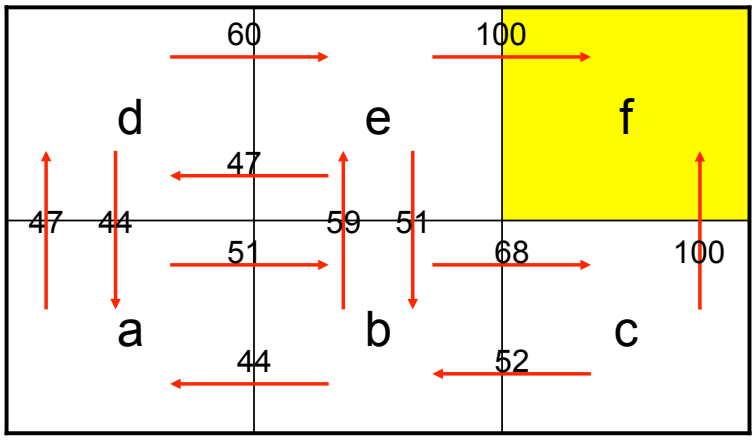
$\gamma = 1$ →

100	100	0
100	100	100

$\gamma = 0.9$ →

52	66	0
49	57	76

Example 1- Value Function Q



$\gamma = 0.9$

Example 2 - Value Function V

5.3	14.4	26.7	
-2.5		-36.6	
-11	-21.9	-37.6	-66.9

π - random

$r = 0 \quad \gamma = 0.9$

$r = 100 \Rightarrow \quad j \to k$

$r = -100 \Rightarrow \quad f \to g \quad d \to g$

Example 2 - Value Function V

-128	-94	-37	
-145		-96	
-151	-145	-128	-117

π - random

$r = -20 \quad \gamma = 0.9$

Bellman Equation for a Policy π

The basic idea:

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

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

Or, without the expectation operator:

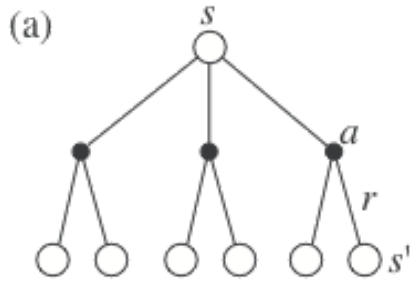
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

Bellman Equation for a Policy π

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P^{a}_{ss'} \cdot \left[R^{a}_{ss'} + \gamma \cdot V^{\pi}(s') \right]$$

The Bellman equation expresses a relationship between the value of a state s and the values of its successor states s'.

Backup diagram:



for V^{π}

Bellman Equation

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P^{a}_{ss'} \cdot [R^{a}_{ss'} + \gamma \cdot V^{\pi}(s')]$$

d	e	f
a	b	c

$$V^{\pi}(a) = \frac{1}{2}\gamma(V^{\pi}(b) + V^{\pi}(d))$$

$$V^{\pi}(e) = \frac{1}{3}\gamma(V^{\pi}(d) + V^{\pi}(b)) + \frac{1}{3}(100 + \gamma \cdot V^{\pi}(f))$$

Bellman Equation for Q and V

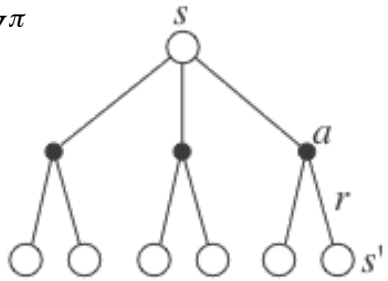
$$Q^{\pi}(s,a) = \sum_{s' \in S} P^{a}_{ss'} \cdot [R^{a}_{ss'} + \gamma \cdot \sum_{a' \in A(s')} \pi(s',a') \cdot Q^{\pi}(s',a')]$$

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P^{a}_{ss'} \cdot [R^{a}_{ss'} + \gamma \cdot V^{\pi}(s')]$$

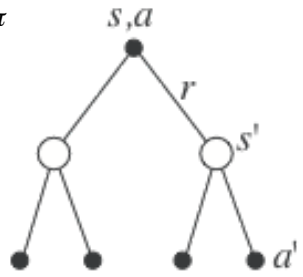
This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

Backup diagrams:

for V^{π}

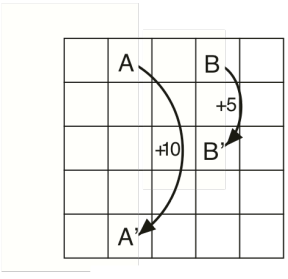


for Q^{π}



Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- Action that takes agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



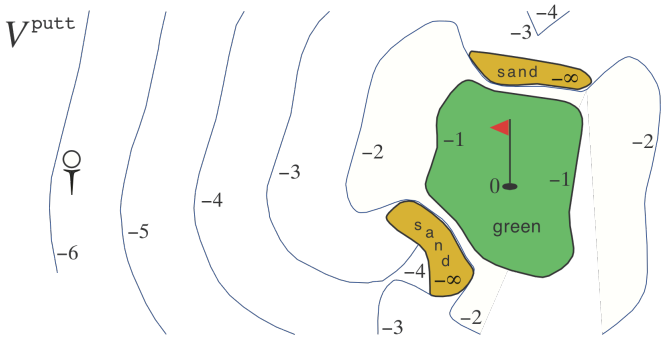
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

State-value function for equiprobable random policy; $\gamma = 0.9$

- Notice the negative values near the lower edge; these are the result of the high probability of hitting the edge of the grid there under the random policy.
- State A is the best state to be in under this policy, but its expected return is less than 10, because from A the agent is taken to A', from which it is likely to run into the edge of the grid.
- State B, is valued more than 5 because from B the agent is taken to B'.

Golf

- State is ball location
- Reward of –1 for each stroke until the ball is in the hole
- Value of a state?
- Actions:
  - `putt` (use putter)
  - `driver` (use driver)
- `putt` succeeds anywhere on the green



State-value function for using the putter

Optimal Value Functions

For finite MDPs, policies can be **partially ordered**: A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of for all states:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \forall\, s \in S$$

There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an **optimal policy** π^*.

Optimal policies share the same **optimal state-value function**:

$$V^*(s) := V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \quad \forall \quad s \in S$$

Optimal policies also share the same **optimal action-value function**:

$$Q^*(s,a) := Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall \quad s \in S,\ a \in A(s)$$

Optimal Policy and Optimal Value Functions

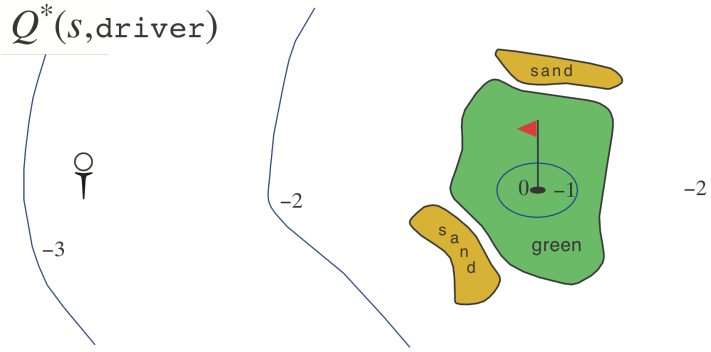
$$Q^*(s,a) := Q^{\pi^*}(s,a) = \max_\pi Q^\pi(s,a) \quad \forall \; s \in S,\, a \in A(s)$$

describes the expected return for taking action *a* in state *s* and thereafter following an optimal policy. Thus, we can write as well:

$$Q^*(s,a) = E\left\{ r_{t+1} + \gamma V^*\left(s_{t+1}\right) \middle| \, s_t = s, a_t = a \right\}$$

Optimal Value Function for Golf

- We can hit the ball farther with `driver` than with `putter`, but with less accuracy
- Q*(*s*,`driver`) gives the value for using `driver` first, then using whichever actions are best



We can reach the hole in one shot using the driver only if we are already very close; thus the -1 contour covers only a small portion of the green.

If we have two strokes we can reach the hole from much farther away, as shown by the -2 contour. In this case we don't have to drive all the way to within the small -1 contour, but only to anywhere on the green; from there we can use the putter. The -3 contour is still farther out and includes the starting tee. From the tee, the best sequence of actions is two drives and one putt, sinking the ball in three strokes.

Bellman Optimality Equation for V^*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

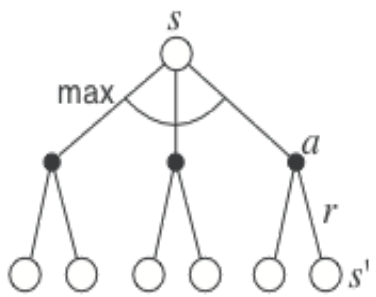
$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_{a \in A(s)} E_{\pi^*}\left\{ R_t \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} E_{\pi^*}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} E_{\pi^*}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} \sum_{s'} \boldsymbol{P}^a_{ss'} \left[\boldsymbol{R}^a_{ss'} + \gamma V^*(s') \right]
\end{aligned}
$$

Bellman Optimality Equation for V^*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^*(s') \right]
\end{aligned}
$$

The relevant backup diagram:



Bellman Optimality Equation for *V**

For finite MDPs, the Bellman optimality equation for V* has a unique solution independent of the policy. The Bellman optimality equation is a system of equations, one for each state, so if there are N states, then there are N equations in unknowns. If the dynamics of the environment are known ($\mathbf{P}^a_{ss'}$ and $\mathbf{R}^a_{ss'}$), then in principle one can solve this system of equations using methods for solving systems of nonlinear equations.

Once one has V*, it is relatively easy to determine an optimal policy. For each state *s*, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy.

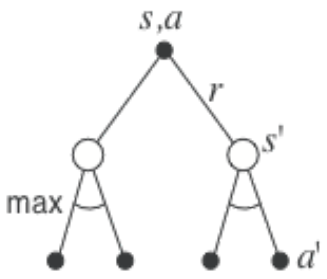
The beauty of V* is that if one uses it to evaluate the short-term consequences of actions - specifically, the one-step consequences - then a greedy policy is actually optimal in the long-term sense in which we are interested because V* already takes into account the reward consequences of all possible future behavior. By means of V*, the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step-ahead search yields the long-term optimal actions.

Bellman Optimality Equation for *Q**

$$Q^*(s,a) = E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') \,\middle|\, s_t = s, a_t = a \right\}$$

$$= \sum_{s'} \boldsymbol{P}^a_{ss'} \left[\boldsymbol{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a') \right]$$

The relevant backup diagram.



Bellman Optimality Equation for Q*

Having Q* makes choosing optimal actions still easier. With Q*, the agent does not even have to do a one-step-ahead search: for any state *s*, it can simply find any action that maximizes Q*(*s*,*a*). The action-value function effectively caches the results of all one-step-ahead searches. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

Bellman Optimality Equation

$$V^*(s) = \max_{a \in A(s)} \sum_{s' \in S} P^a_{ss'} \cdot [R^a_{ss'} + \gamma \cdot V^*(s')]$$

$$Q^*(s,a) = \sum_{s' \in S} P^a_{ss'} \cdot [R^a_{ss'} + \gamma \cdot \max_{a' \in A(s')} Q^*(s',a')]$$

d	e	f
a	b	c

$$V^*(s) = \max_{a \in A(s)} \{r(s,a) + \gamma \cdot V^*(s(a))\}$$

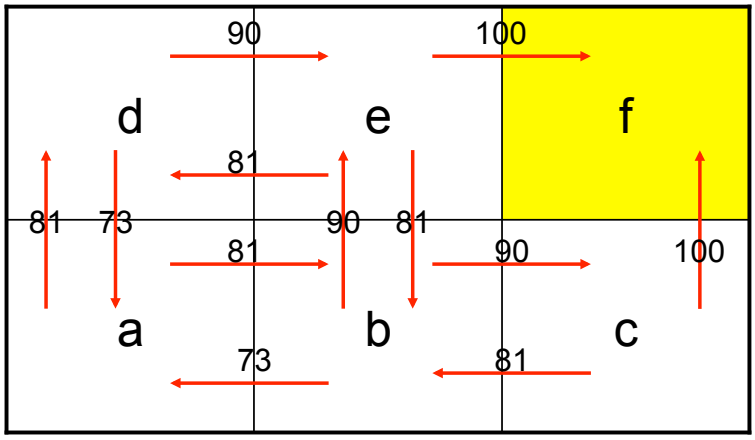
$$V^*(a) = \max \{\gamma \cdot V^*(b), \gamma \cdot V^*(d)\}$$

Example 1

$\gamma = 0.9$

90	100	f
81	90	100

Example 1



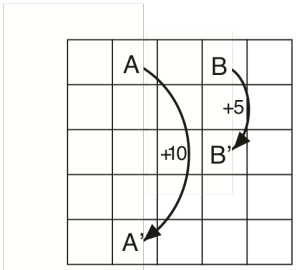
$\gamma = 0.9$

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

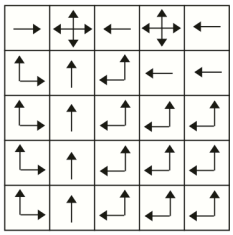
E.g., back to the gridworld:



a) 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

b) V^*



c) π^*

What About Optimal Action-Value Functions?

Given Q^* , the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - accurate knowledge of environment dynamics;
  - we have enough space and time to do the computation;
  - the Markov Property.
- How much space and time do we need?
  - A solution requires an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards.
  - BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Summary

- Agent-environment interaction
  - States
  - Actions
  - Rewards
- Policy: stochastic rule for selecting actions
- Return: the function of future rewards agent tries to maximize
- Episodic and continuing tasks
- Markov Property
- Markov Decision Process
  - Transition probabilities
  - Expected rewards
- Value functions
  - State-value function for a policy
  - Action-value function for a policy
  - Optimal state-value function
  - Optimal action-value function
- Optimal value functions
- Optimal policies
- Bellman Equations
- The need for approximation