COMP 532
Machine Learning and BioInspired Optimization

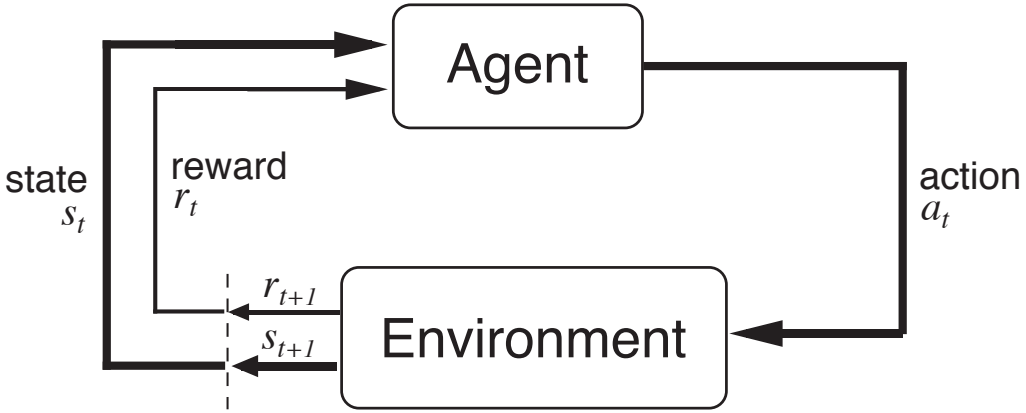
Lecture 5 Reinforcement Learning

Dr. Shan Luo

Department of Computer Science

shan.luo@liverpool.ac.uk

The Agent-Environment Interface



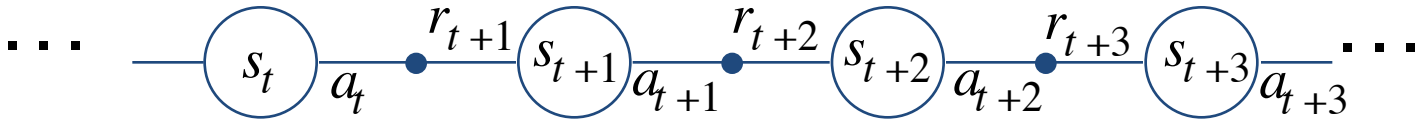
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \mathrm{K}$

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}



The Agent Learns a Policy

Policy at step t, π_t:

a mapping from states to action probabilities

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

- 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.

The reward hypothesis

That all of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward)

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**, $E\{Rt\}$, for each step t.

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.

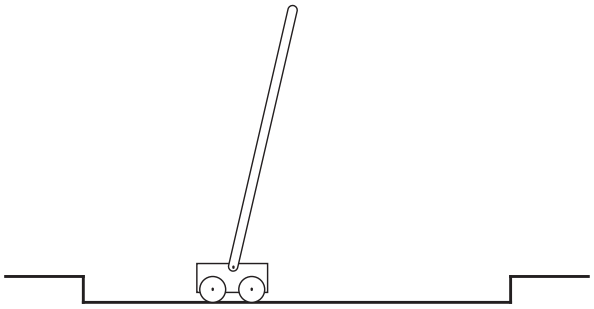
Discounted return:

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

where $0 \leq \gamma \leq 1$ 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:

$$\text{reward} = +1 \text{ for each step before failure}$$

$$\Rightarrow \text{return} = \text{number of steps before failure}$$

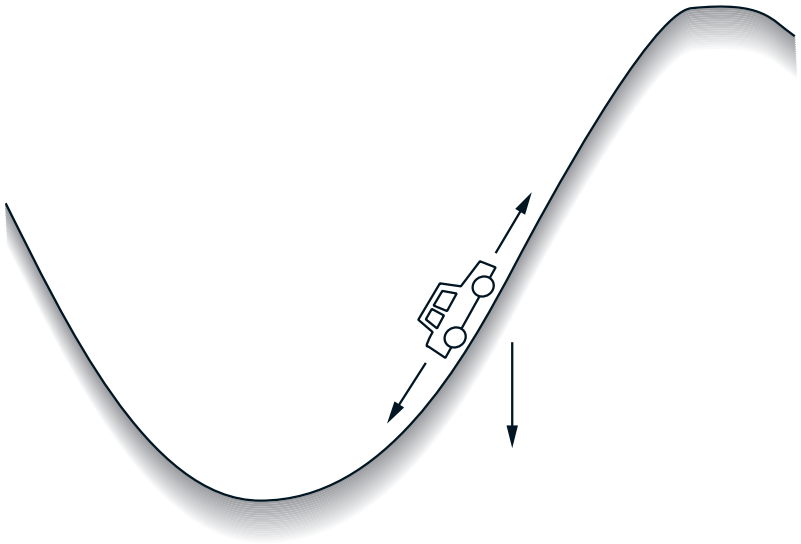
As a **continuing task** with discounted return:

$$\text{reward} = -1 \text{ upon failure; } 0 \text{ otherwise}$$

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

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.

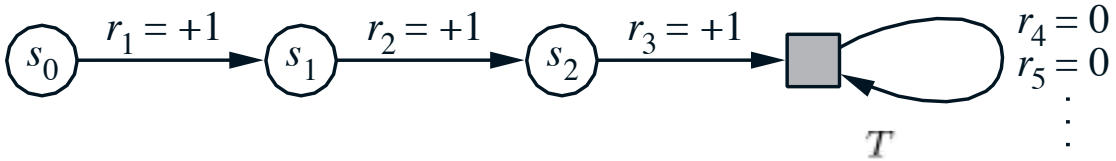
reward = −1 for each step where **not** at top of hill

$\Rightarrow$ 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

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have to 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 <u>all</u> cases by writing $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$,

  including the possibility that $T = \infty$ or $\gamma = 1$ (not both)

The Markov Property

- By "the state" at step *t*, the book means 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**:

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

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

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**:

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

  - **rewards expectations**:

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

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 it runs out of power while searching, it 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	a	s'	$p(s'\|s,a)$	$r(s,a,s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	$r_{\texttt{wait}}$
low	wait	high	0	$r_{\texttt{wait}}$
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	0.

Recycling Robot MDP

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

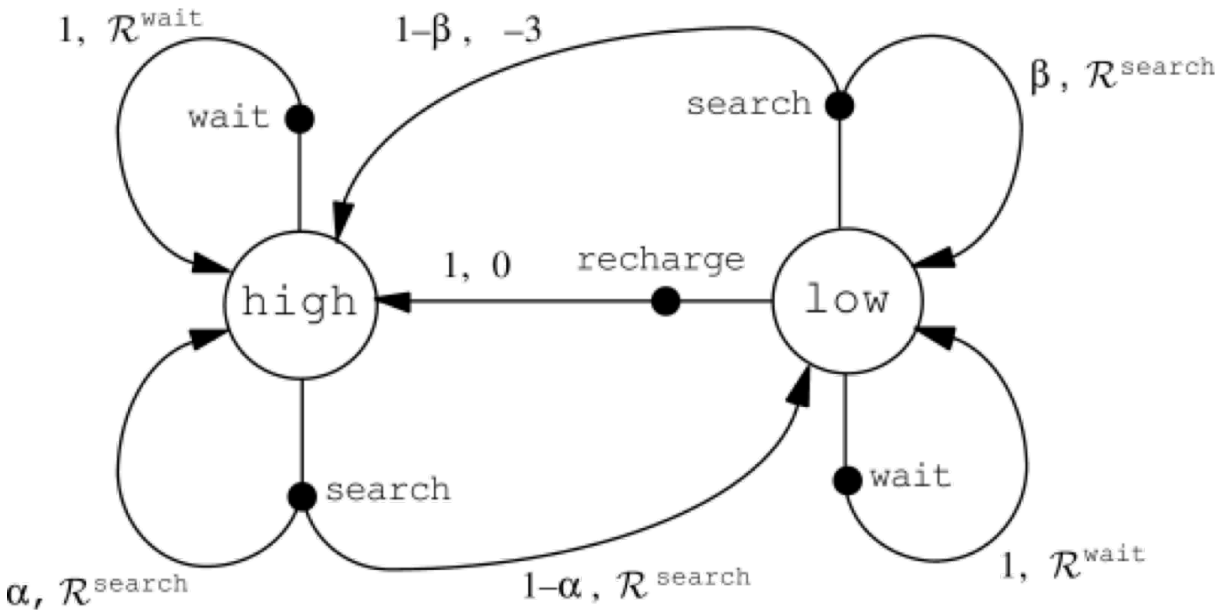
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

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

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

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

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



Value Functions

- The **value of a state** is the expected return starting from that state; 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:

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

Bellman Equation for a Policy π

The basic idea:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

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

$$= r_{t+1} + \gamma R_{t+1}$$

So:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \,\middle|\, s_t = s\right\}$$

$$= E_{\pi}\left\{r_{t+1} + \gamma V\left(s_{t+1}\right) \,\middle|\, s_t = s\right\}$$

Or, without the expectation operator:

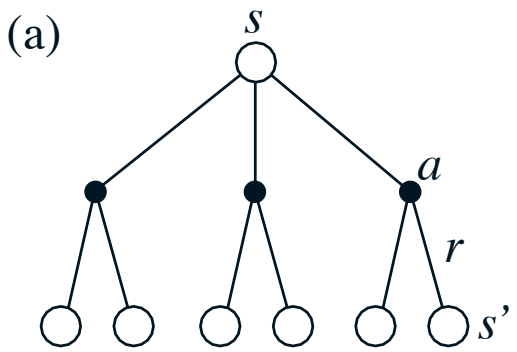
$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathbf{P}^a_{ss'}\left[\mathbf{R}^a_{ss'} + \gamma V^{\pi}(s')\right]$$

More on the Bellman Equation

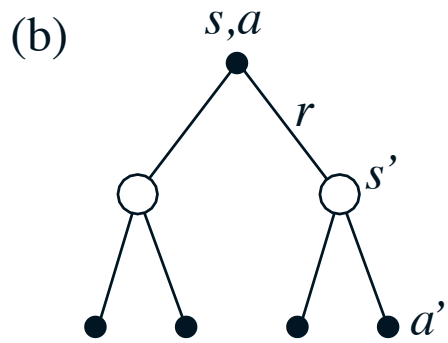
$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathbf{P}^{a}_{ss'} \left[\mathbf{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

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}

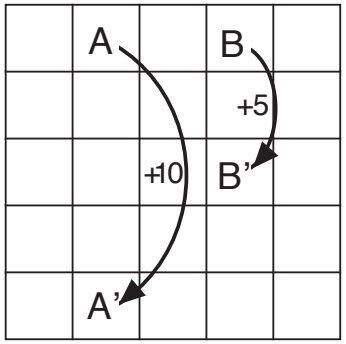
Bellman Equation: Intuition

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

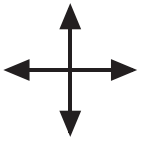
The equations express the recursive relation between the value of a state and its successor states and it averages over all the possibilities, weighting each by its probability of occurring.

Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take 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.



(a)



Actions

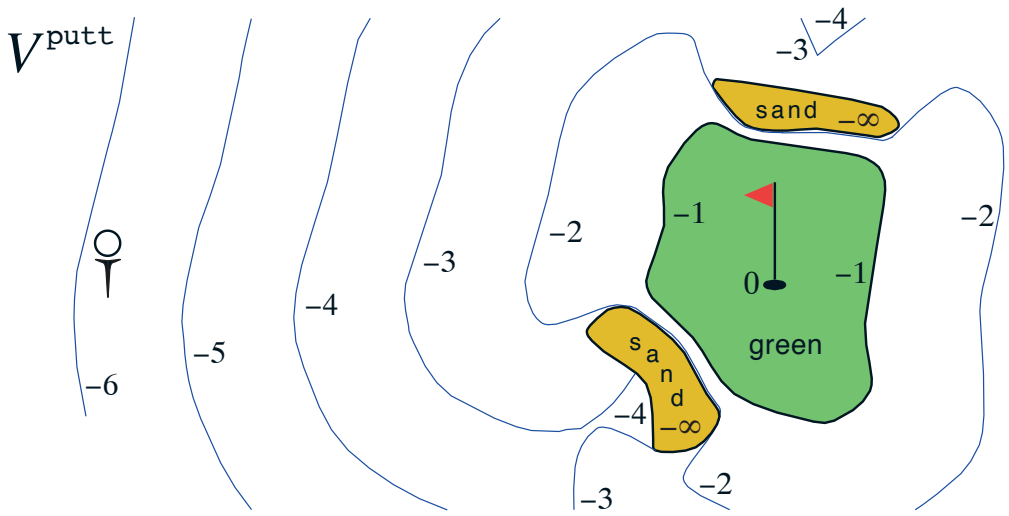
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

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

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



Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

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

- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π^*.
- Optimal policies share the same **optimal state-value function**:

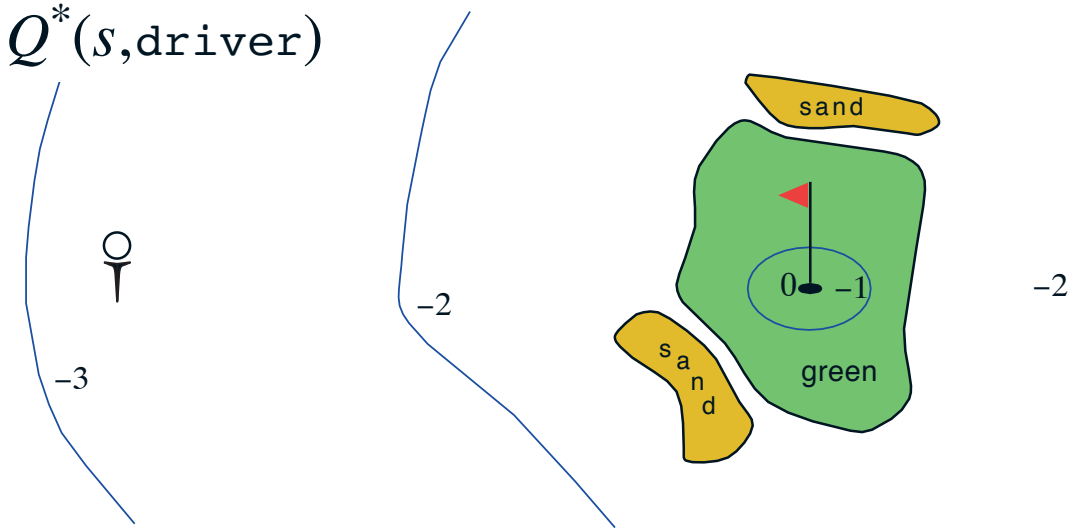
$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$$

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

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

Optimal Value Function for Golf

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

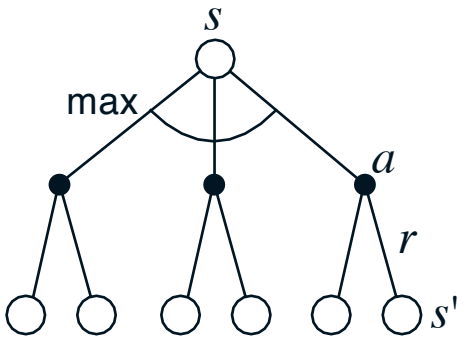


Bellman Optimality Equation for V^*

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:



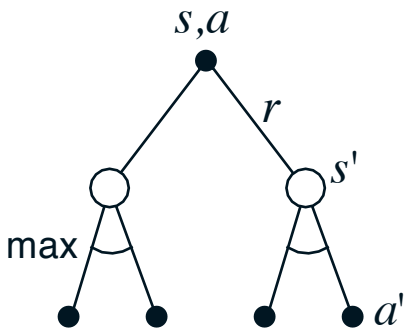
V^* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for Q^*

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

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

The relevant backup diagram:



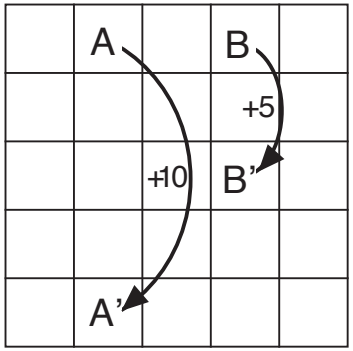
Q^* is the unique solution of this system of nonlinear equations.

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.

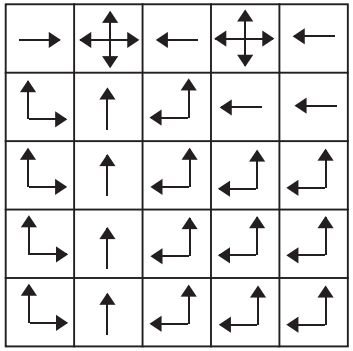
E.g., back to the gridworld:



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:

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

Recap: Bellman Optimality Equation

- Optimal value function:

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

- Analogous for Q^*(optimal action-value):

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

$$V^*(s) = \max_{a} Q^{\pi^*}(s,a)$$

Solving the Bellman Equation

- Finding optimal policy by explicitly solving Bellman: **Dynamic Programming**

- Rarely useful in practice, it requires:
  - accurate knowledge of environment dynamics;
  - enough space and time to do the computation;
  - the Markov Property.

Solving the Bellman Optimality Equation

- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods; Chapter 4),
  - 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.

Value Iteration

Possibility to finding optimal policy

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Bootstrapping!

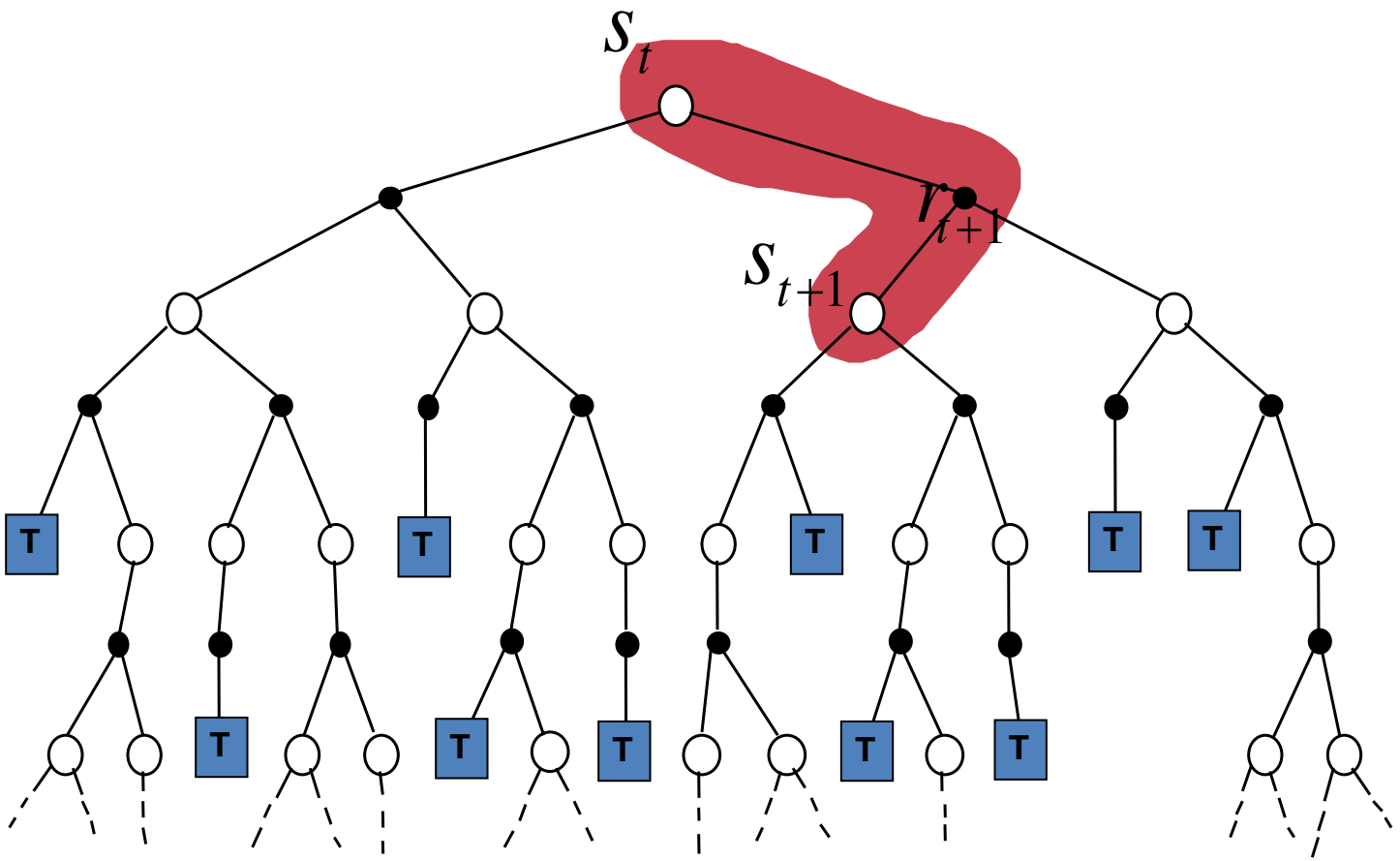
Value Iteration: Temporal Difference

No model of the environment

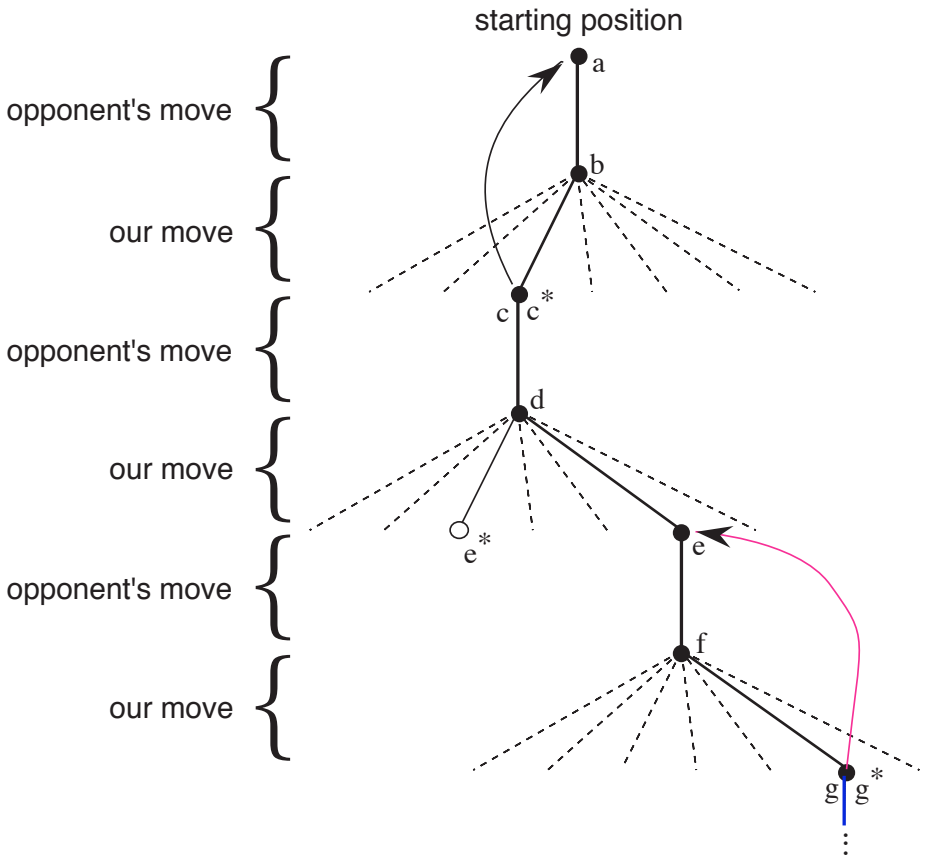
- However, the update rule requires to know the dynamics of the environment.
- Typical is to use temporal difference methods to overcome this problem, like Q-learning
- Look at the difference between the current estimate of the value of a state and the discounted value of the next state and the reward received.

Value Iteration: Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



Remember Tic-Tac-Toe?



$$V(a) \leftarrow V(a) + \alpha[V(c) - V(a)]$$

$$V(e) \leftarrow V(e) + \alpha[V(g) - V(e)]$$

Value iteration: $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$

where r_{t+1} can be 0, and γ can be 1 !