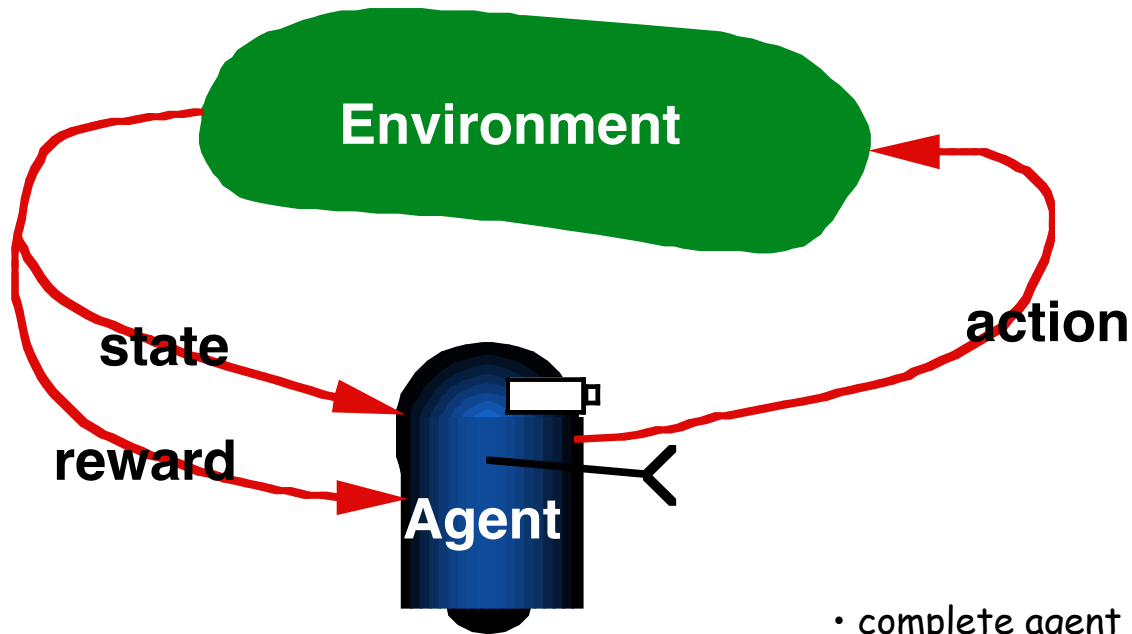


Examples and Videos of Markov Decision Processes (MDPs) and Reinforcement Learning

Artificial Intelligence is
interaction to achieve a goal



- complete agent
- temporally situated
- continual learning & planning
- object is to affect environment
- environment stochastic & uncertain

States, Actions, and Rewards

The RoboCup Soccer Competition



©2002 The RoboCup Federation



©2002 The RoboCup Federation



Cyberoos2000 0:0 Brainstormers_2K play_on 355



©2002 The RoboCup Federation

Autonomous Learning of Efficient Gait

Kohl & Stone (UTexas) 2004



Policies

- A **policy** maps each state to an action to take
 - Like a stimulus–response rule
- We seek a policy that maximizes cumulative reward
- The policy is a subgoal to achieving reward

The Reward Hypothesis

The goal of intelligence is to maximize the cumulative sum of a single received number:

“reward” = pleasure - pain

Artificial Intelligence = reward maximization

Value

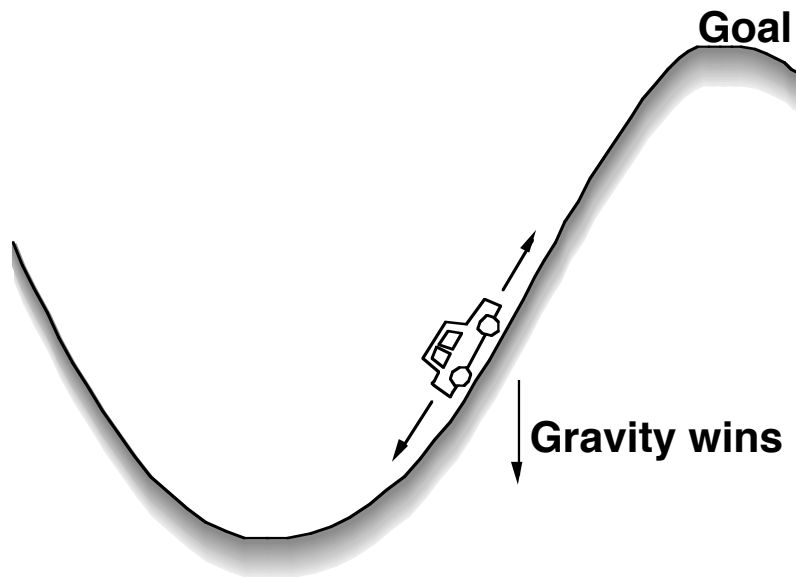
Value systems are hedonism with foresight

We value situations according to how much reward we expect will follow them

All efficient methods for solving sequential decision problems determine (learn or compute) “value functions” as an intermediate step

Value systems are a *means* to reward, yet we *care more* about values than rewards

The Mountain Car Problem



SITUATIONS: car's position and velocity

ACTIONS: three thrusts: forward, reverse, none

REWARDS: always -1 until car reaches the goal

No Discounting

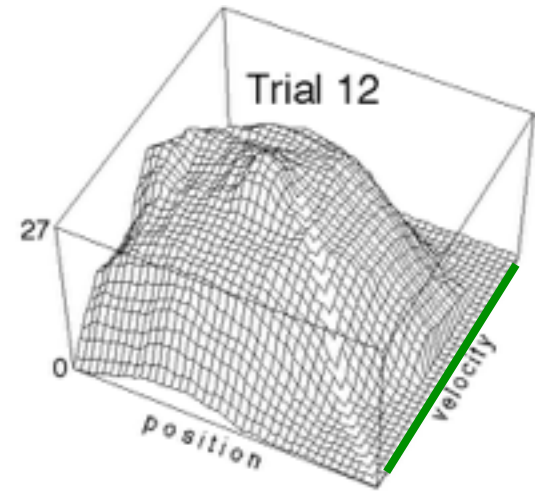
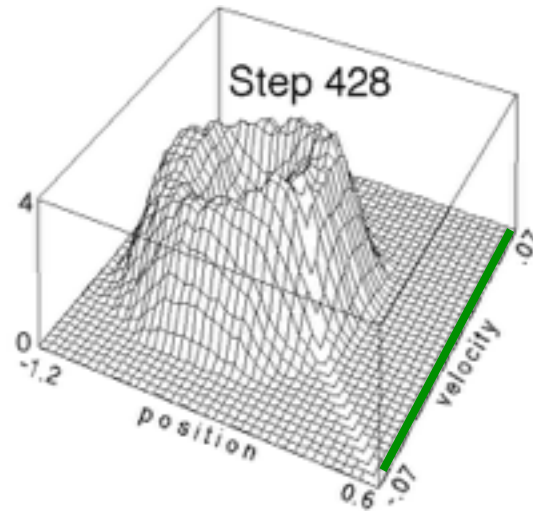
Minimum-Time-to-Goal Problem

Value Functions Learned while solving the Mountain Car problem

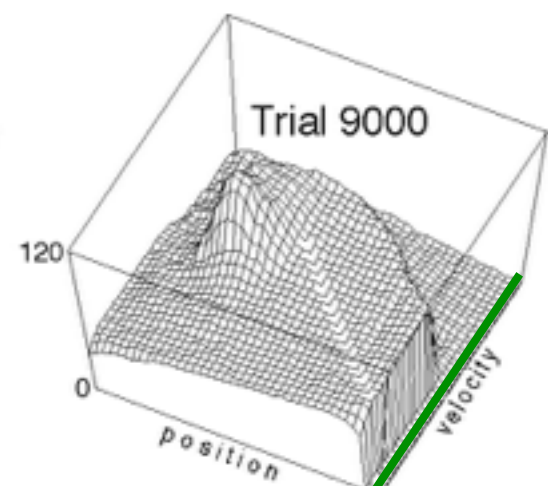
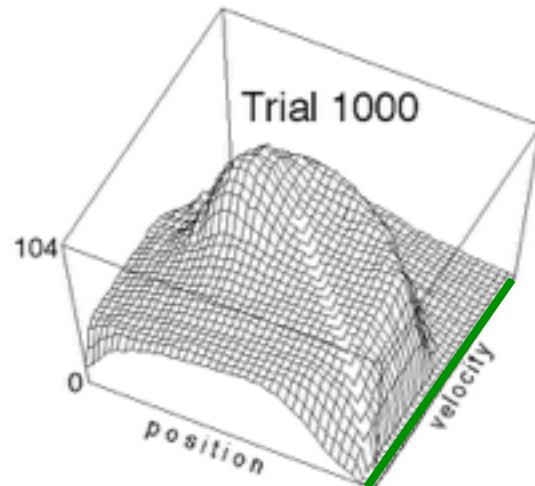
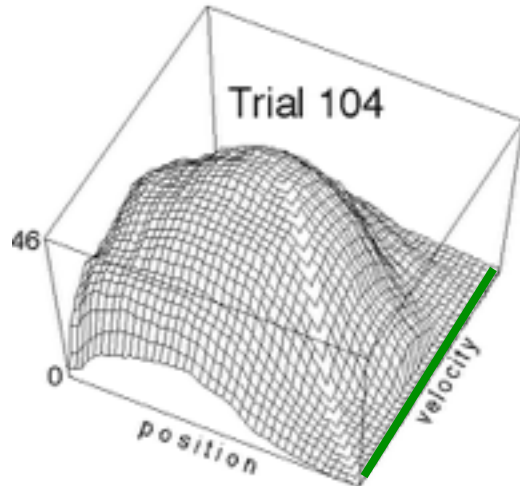


Minimize Time-to-Goal

Value = estimated time to goal



Goal region

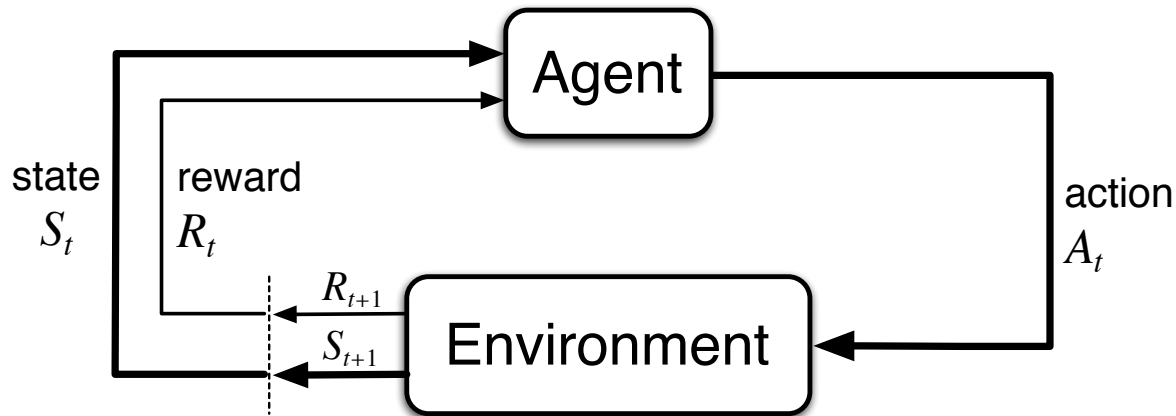


Chapter 3: The Reinforcement Learning Problem (Markov Decision Processes, or MDPs)

Objectives of this chapter:

- present Markov decision processes— an idealized form of the AI problem for which we have precise theoretical results
- introduce key components of the mathematics: value functions and Bellman equations

The Agent-Environment Interface



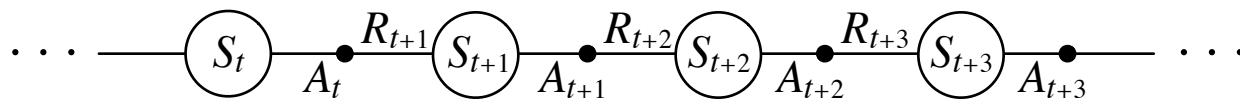
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \dots$

Agent observes state at step t : $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$



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”

$$p(s', r | s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

- there is also:

$$p(s' | s, a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

$$r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

The Agent Learns a Policy

Policy at step $t = \pi_t =$

a mapping from states to action probabilities

$\pi_t(a | s) =$ probability that $A_t = a$ when $S_t = s$

Special case - *deterministic policies*:

$\pi_t(s) =$ the action taken with prob=1 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 Meaning of Life

(goals, rewards, and returns)

Return

Suppose the sequence of rewards after step t is:

$$R_{t+1}, R_{t+2}, R_{t+3}, \dots$$

What do we want to maximize?

At least three cases, but in all of them,

we seek to maximize the **expected return**, $E\{G_t\}$, on each step t .

- Total reward, $G_t =$ sum of all future reward in the episode
- Discounted reward, $G_t =$ sum of all future *discounted* reward
- Average reward, $G_t =$ average reward per time step

Episodic Tasks

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

In episodic tasks, we almost always use simple *total reward*:

$$G_t = R_{t+1} + R_{t+2} + \dots + R_T,$$

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

Continuing Tasks

Continuing tasks: interaction does not have natural episodes, but just goes on and on...

In this class, for continuing tasks we will always use *discounted return*:

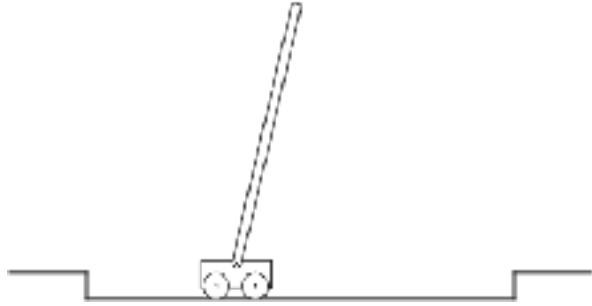
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

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

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

Typically, $\gamma = 0.9$

An Example: Pole Balancing



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

⇒ return = number of steps before failure

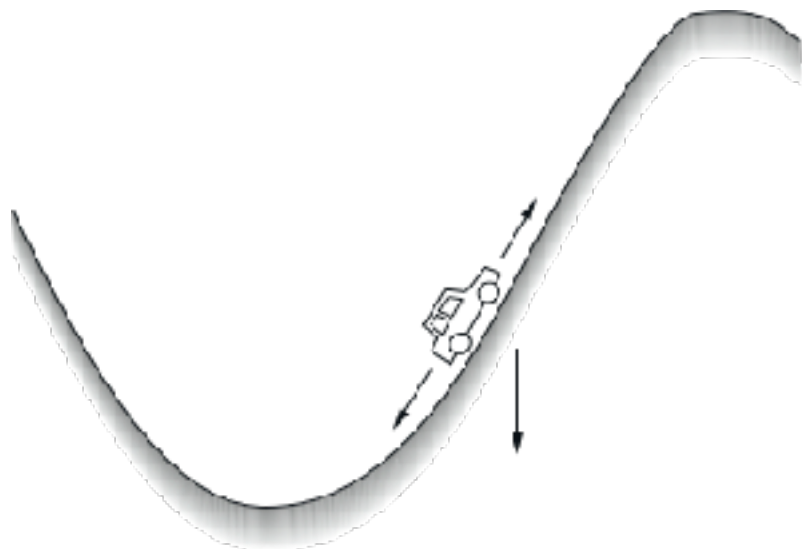
As a **continuing task** with discounted return:

reward = -1 upon failure; 0 otherwise

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

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

Another Example: Mountain Car



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

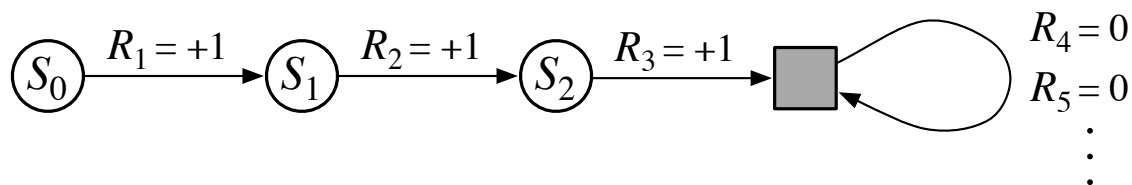
reward = -1 for each step where **not** at top of hill

⇒ 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 Trick to Unify Notation for Returns

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



- ❑ We can cover all cases by writing $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$,

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

Rewards and returns

- The objective in RL is to maximize long-term future reward
- That is, to choose A_t so as to maximize $R_{t+1}, R_{t+2}, R_{t+3}, \dots$
- But what exactly should be maximized?
- The discounted return at time t :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \begin{array}{l} \text{the discount rate} \\ \gamma \in [0, 1) \end{array}$$

γ	Reward sequence	Return
0.5(or any)	1 0 0 0...	1
0.5	0 0 2 0 0 0...	0.5
0.9	0 0 2 0 0 0...	1.62
0.5	-1 2 6 3 2 0 0 0...	2

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 = 26 \quad G_1 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 = 26 \quad G_1 = 26 \quad G_0 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 = 26 \quad G_1 = 26 \quad G_0 = 14$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 = 26 \quad G_1 = 26 \quad G_0 = 14$$

- And if $\gamma = 0.9$?

$$G_1 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 = 26 \quad G_1 = 26 \quad G_0 = 14$$

- And if $\gamma = 0.9$?

$$G_1 = 130 \quad G_0 =$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \gamma \in [0, 1)$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 6, R_3 = -12, R_4 = 16, \text{ then zeros for } R_5 \text{ and later}$$

- What are the following returns?

$$G_4 = 0 \quad G_3 = 16 \quad G_2 = -4 \quad G_1 = 4 \quad G_0 = 3$$

- Suppose $\gamma = 0.5$ and the reward sequence is all 1s.

$$G = \frac{1}{1 - \gamma} = 2$$

- Suppose $\gamma = 0.5$ and the reward sequence is

$$R_1 = 1, R_2 = 13, R_3 = 13, R_4 = 13, \text{ and so on, all 13s}$$

$$G_2 = 26 \quad G_1 = 26 \quad G_0 = 14$$

- And if $\gamma = 0.9$?

$$G_1 = 130 \quad G_0 = 118$$

4 value functions

	state values	action values
prediction	v_{π}	q_{π}
control	v_{*}	q_{*}

- All theoretical objects, mathematical ideals (expected values)
- Distinct from their estimates:

$$V_t(s) \quad Q_t(s, a)$$

Values are *expected* returns

- The value of a state, given a policy:

$$v_\pi(s) = \mathbb{E}\{G_t \mid S_t = s, A_{t:\infty} \sim \pi\} \quad v_\pi : \mathcal{S} \rightarrow \mathbb{R}$$

- The value of a state-action pair, given a policy:

$$q_\pi(s, a) = \mathbb{E}\{G_t \mid S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\} \quad q_\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

- The optimal value of a state:

$$v_*(s) = \max_{\pi} v_\pi(s) \quad v_* : \mathcal{S} \rightarrow \mathbb{R}$$

- The optimal value of a state-action pair:

$$q_*(s, a) = \max_{\pi} q_\pi(s, a) \quad q_* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

- Optimal policy: π_* is an optimal policy if and only if

$$\pi_*(a|s) > 0 \text{ only where } q_*(s, a) = \max_b q_*(s, b) \quad \forall s \in \mathcal{S}$$

- in other words, π_* is optimal iff it is *greedy* wrt q_*

4 value functions

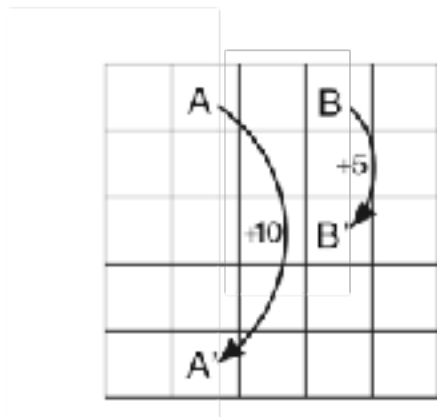
	state values	action values
prediction	v_{π}	q_{π}
control	v_{*}	q_{*}

- All theoretical objects, mathematical ideals (expected values)
- Distinct from their estimates:

$$V_t(s) \quad Q_t(s, a)$$

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;
 $\gamma = 0.9$

What we learned so far

- ❑ Finite Markov decision processes!
 - States, actions, and rewards
 - And returns
 - And time, discrete time
 - They capture essential elements of life — state, causality
- ❑ The goal is to optimize expected returns
 - returns are *discounted sums* of *future* rewards
- ❑ Thus we are interested in *values* — expected returns
- ❑ There are four value *functions*
 - state vs state-action values
 - values for a policy vs values for the optimal policy

Optimal Value Functions

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

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

- 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**:

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

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

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

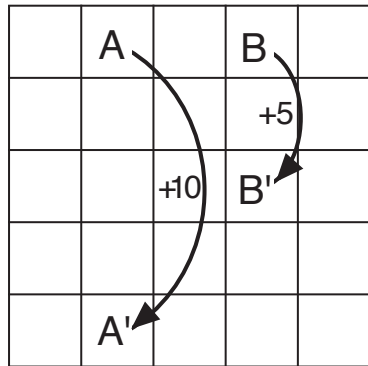
This is the expected return for taking action a in state s and thereafter following an optimal policy.

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.

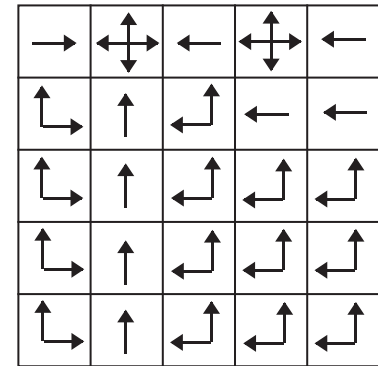
E.g., back to the gridworld:



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:

$$\pi_*(s) = \arg \max_a q_*(s, a)$$

Value Functions

x 4

Bellman Equations

x 4

Bellman Equation for a Policy π

The basic idea:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

So:

$$\begin{aligned} v_\pi(s) &= E_\pi \{ G_t | S_t = s \} \\ &= E_\pi \{ R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \} \end{aligned}$$

Or, without the expectation operator:

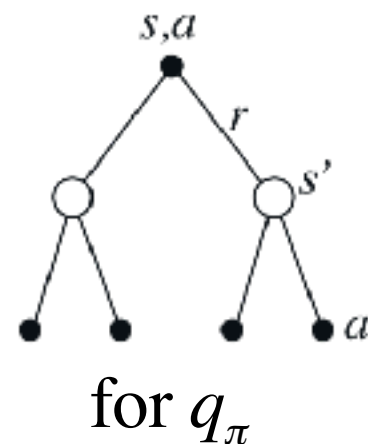
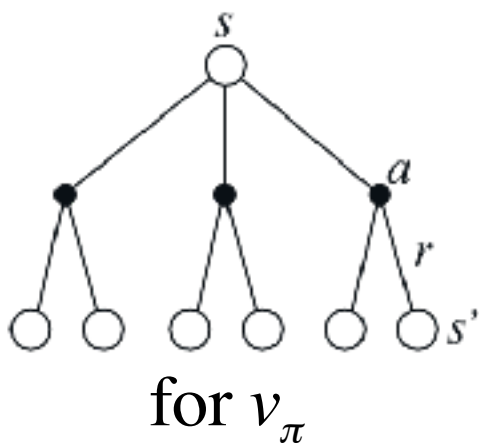
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right]$$

More on the Bellman Equation

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

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

Backup diagrams:

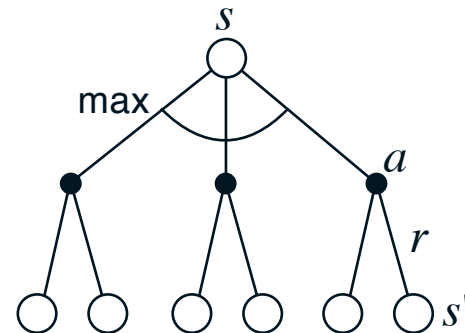


Bellman Optimality Equation for v_*

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

$$\begin{aligned}v_*(s) &= \max_a q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')].\end{aligned}$$

The relevant backup diagram:

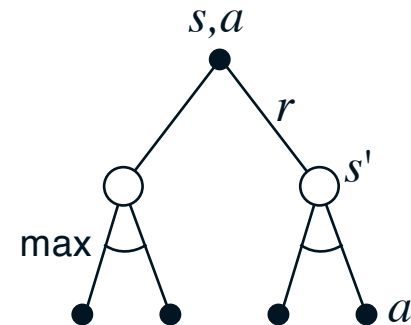


v_* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for q_*

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]. \end{aligned}$$

The relevant backup diagram:



q_* is the unique solution of this system of nonlinear equations.

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?
 - 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.

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

Chapter 4: Dynamic Programming

Objectives of this chapter:

- ❑ Overview of a collection of classical solution methods for MDPs known as dynamic programming (DP)
- ❑ Show how DP can be used to compute value functions, and hence, optimal policies
- ❑ Discuss efficiency and utility of DP

Policy Evaluation (Prediction)

Policy Evaluation: for a given policy π , compute the state-value function v_π

Recall: **State-value function for policy π**

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$


Recall: **Bellman equation for v_π**

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

—a system of $|S|$ simultaneous equations

Iterative Policy Evaluation (Prediction)

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

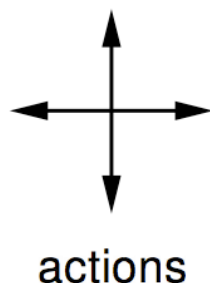
a “sweep” 

A sweep consists of applying a **backup operation** to each state.

A full policy-evaluation backup:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

A Small Gridworld Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

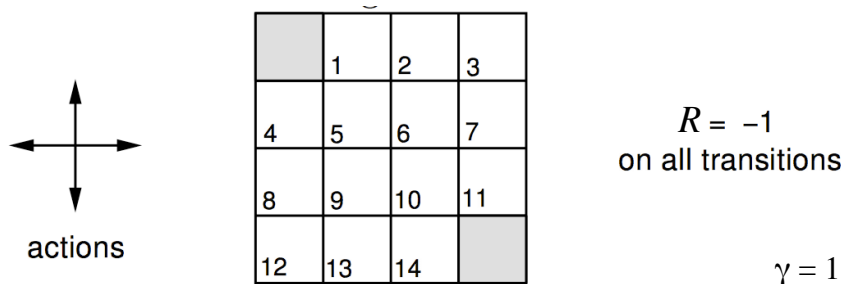
$$\gamma = 1$$

- ❑ An undiscounted episodic task
- ❑ Nonterminal states: 1, 2, . . . , 14;
- ❑ One terminal state (shown twice as shaded squares)
- ❑ Actions that would take agent off the grid leave state unchanged
- ❑ Reward is -1 until the terminal state is reached

Iterative Policy Eval for the Small Gridworld

V_k for the
Random Policy

$\pi =$ equiprobable random action choices



$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

- An undiscounted episodic task
- Nonterminal states: 1, 2, ..., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

Iterative Policy Evaluation – One array version

Input π , the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Enough Prediction,
let's start towards Control!

Policy improvement theorem

- Given the value function for *any policy* π :

$$q_{\pi}(s, a) \quad \text{for all } s, a$$

- It can always be **greedified** to obtain a *better policy*:

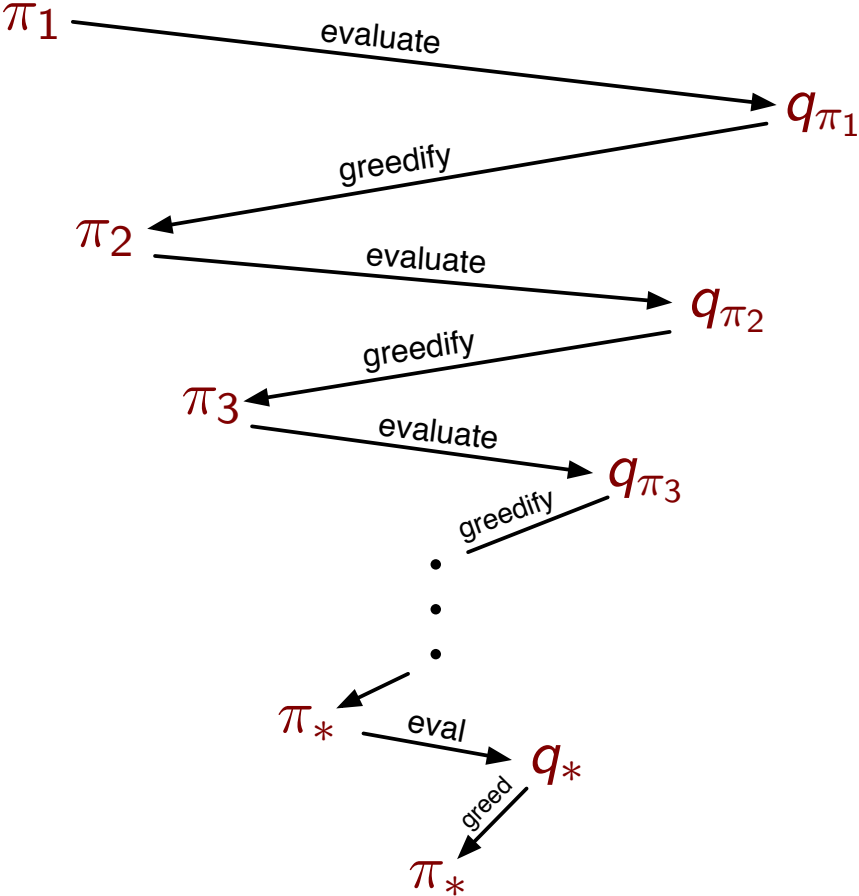
$$\pi'(s) = \arg \max_a q_{\pi}(s, a) \quad (\pi' \text{ is not unique})$$

- where better means:

$$q_{\pi'}(s, a) \geq q_{\pi}(s, a) \quad \text{for all } s, a$$

- with equality only if both policies are optimal

The dance of policy and value (Policy Iteration)



Any policy evaluates to a unique value function (soon we will see how to learn it)

which can be greedified to produce a better policy

That in turn evaluates to a value function

which can in turn be greedified...

Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

The dance converges in a *finite number of steps*, usually very few

Policy Improvement

Suppose we have computed v_π for a deterministic policy π .

For a given state s ,

would it be better to do an action $a \neq \pi(s)$?

And, we can compute $q_\pi(s, a)$ from v_π by:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]. \end{aligned}$$

Policy Improvement

Suppose we have computed v_π for a deterministic policy π .

For a given state s ,

would it be better to do an action $a \neq \pi(s)$?

It is better to switch to action a for state s if and only if

$$q_\pi(s, a) > v_\pi(s)$$

And, we can compute $q_\pi(s, a)$ from v_π by:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]. \end{aligned}$$

Policy Improvement Cont.

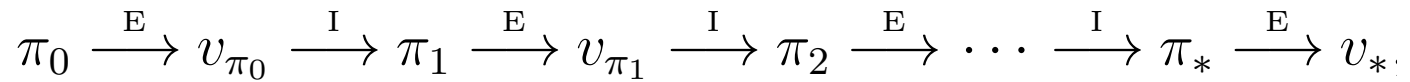
Do this for all states to get a new policy $\pi' \geq \pi$ that is **greedy** with respect to v_π :

$$\begin{aligned}\pi'(s) &= \arg \max_a q_\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')],\end{aligned}$$

What if the policy is unchanged by this?

Then the policy must be optimal!

Policy Iteration

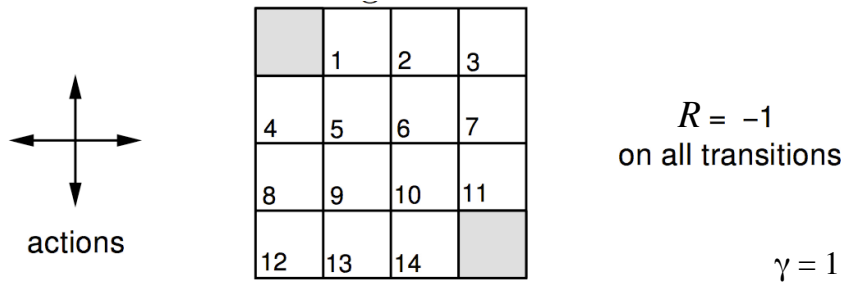


policy evaluation

policy improvement
“greedification”

Iterative Policy Eval for the Small Gridworld

$\pi =$ equiprobable random action choices



- ❑ An undiscounted episodic task
- ❑ Nonterminal states: 1, 2, ..., 14;
- ❑ One terminal state (shown twice as shaded squares)
- ❑ Actions that would take agent off the grid leave state unchanged
- ❑ Reward is -1 until the terminal state is reached

$$\pi'(s) \doteq \arg \max_a \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_{\pi}(s') \right]$$

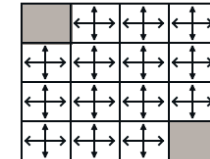
for all $s \in \mathcal{S}$

V_k for the
Random Policy

Greedy Policy
w.r.t. V_k

$k = 0$

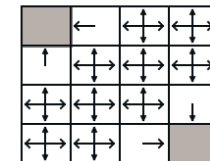
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random
policy

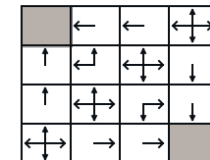
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



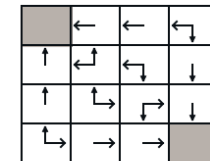
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



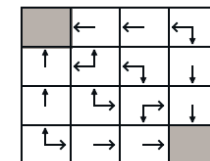
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



$k = 10$

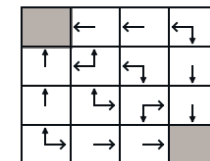
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



optimal
policy

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Iteration – One array version (+ policy)

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

policy-stable \leftarrow *true*

For each $s \in \mathcal{S}$:

$a \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$

If $a \neq \pi(s)$, then *policy-stable* \leftarrow *false*

If *policy-stable*, then stop and return V and π ; else go to 2

Value Iteration

Recall the **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

Here is the **full value-iteration backup**:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

Value Iteration – One array version

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

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π , such that

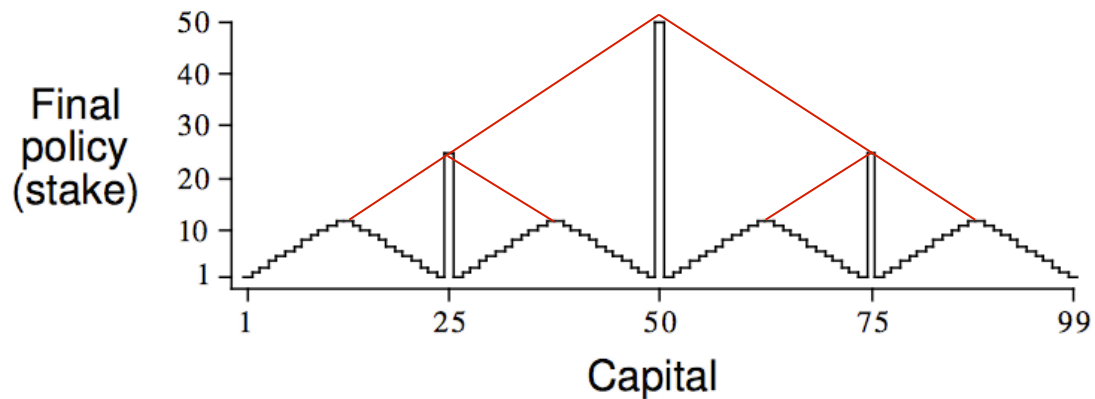
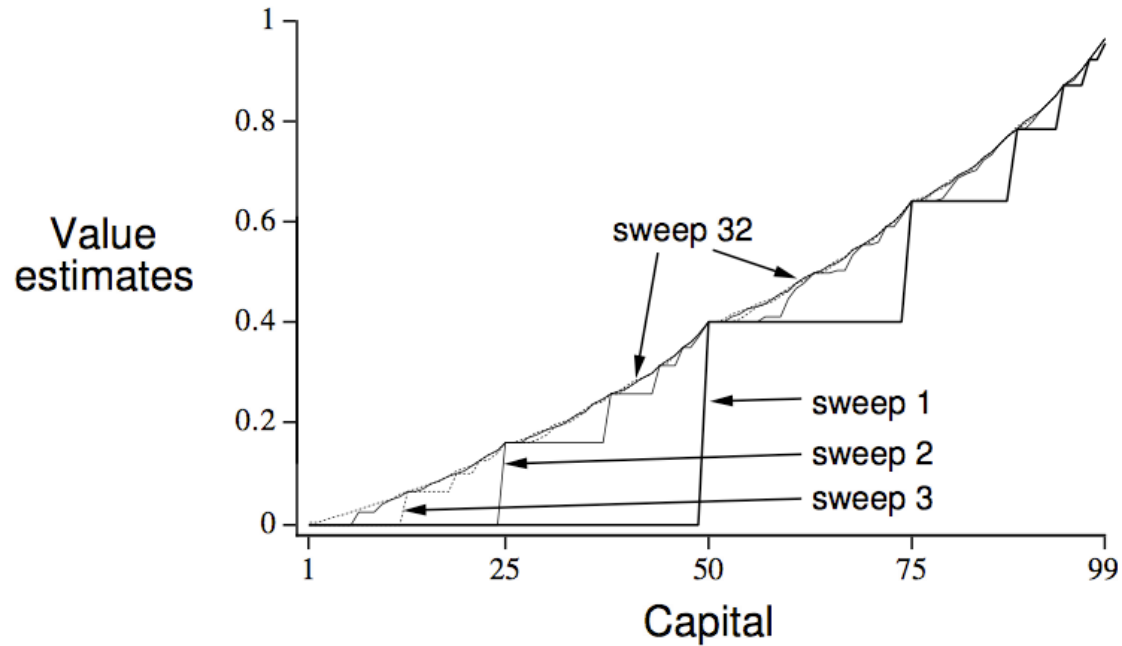
$$\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

Gambler's Problem

- ❑ Gambler can repeatedly bet \$ on a coin flip
- ❑ Heads he wins his stake, tails he loses it
- ❑ Initial capital $\in \{\$1, \$2, \dots \$99\}$
- ❑ Gambler wins if his capital becomes \$100
loses if it becomes \$0
- ❑ Coin is unfair
 - Heads (gambler wins) with probability $p = .4$

- ❑ States, Actions, Rewards? Discounting?

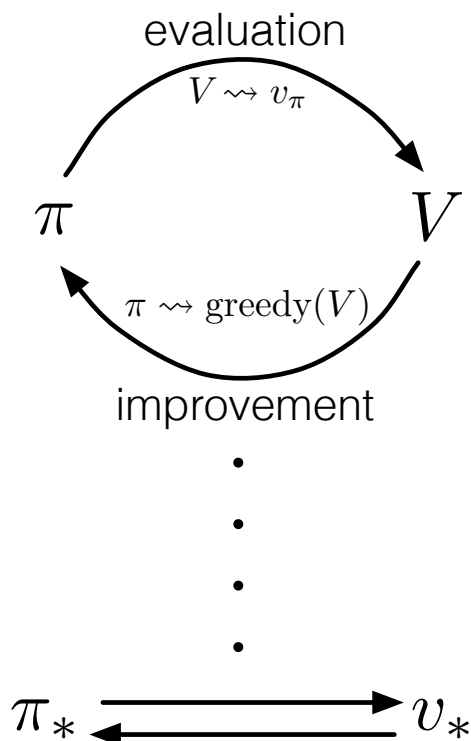
Gambler's Problem Solution



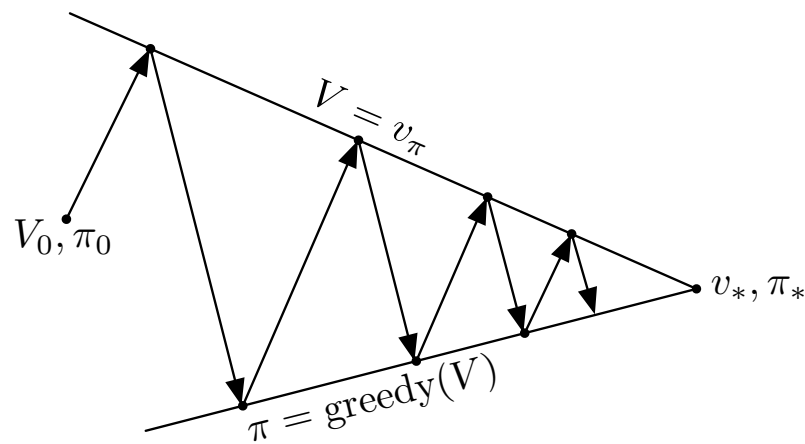
Generalized Policy Iteration

Generalized Policy Iteration (GPI):

any interaction of policy evaluation and policy improvement, independent of their granularity.



A geometric metaphor for convergence of GPI:



Asynchronous DP

- ❑ All the DP methods described so far require exhaustive sweeps of the entire state set.
- ❑ Asynchronous DP does not use sweeps. Instead it works like this:
 - Repeat until convergence criterion is met:
 - Pick a state at random and apply the appropriate backup
- ❑ Still need lots of computation, but does not get locked into hopelessly long sweeps
- ❑ Can you select states to backup intelligently? YES: an agent's experience can act as a guide.

Efficiency of DP

- ❑ To find an optimal policy is polynomial in the number of states...
- ❑ BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- ❑ In practice, classical DP can be applied to problems with a few millions of states.
- ❑ Asynchronous DP can be applied to larger problems, and is appropriate for parallel computation.
- ❑ It is surprisingly easy to come up with MDPs for which DP methods are not practical.

Summary

- ❑ Policy evaluation: backups without a max (prediction)
- ❑ Policy improvement: form a greedy policy, if only locally
- ❑ Policy iteration: alternate the above two processes (control)
- ❑ Value iteration: backups with a max (control)

- ❑ Full backups (to be contrasted later with sample backups)
- ❑ Generalized Policy Iteration (GPI)
- ❑ Asynchronous DP: a way to avoid exhaustive sweeps
- ❑ **Bootstrapping**: updating estimates based on other estimates
- ❑ Biggest limitation of DP is that it requires a *probability model* (as opposed to a generative or simulation model)

Chapter 6: Temporal Difference Learning

Objectives of this chapter:

- Introduce Temporal Difference (TD) learning
- Focus first on policy evaluation, or prediction, methods
- Then extend to control methods

TD methods bootstrap and sample

- ▶ Bootstrapping: update involves an estimate of the value function
 - TD and DP methods bootstrap
- ▶ Sampling: update **does not** involve an expected value
 - TD methods sample
 - Classical DP **does not** sample

The Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx q_*(a), \quad \forall a \quad \text{action-value estimates}$$

- Define the *greedy action* at time t as

$$A_t^* \doteq \arg \max_a Q_t(a)$$

- If $A_t = A_t^*$ then you are *exploiting*
If $A_t \neq A_t^*$ then you are *exploring*
- You can't do both, but you need to do both
- You can never stop exploring, but maybe you should explore less with time. Or maybe not.

ϵ -Greedy Action Selection

- In greedy action selection, you always exploit
- In ϵ -greedy, you are usually greedy, but with probability ϵ you instead pick an action at random (possibly the greedy action again)
- This is perhaps the simplest way to balance exploration and exploitation

TD Prediction

Policy Evaluation (the prediction problem):

for a given policy π , compute the state-value function v_π

The simplest temporal-difference method TD(0):

$$V(S_t) = V(S_t) + \alpha \left[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target}} - V(S_t) \right]$$

target: an estimate of the return

TD target for prediction

- ▶ The TD target: $R_{t+1} + \gamma v_{\pi}(S_{t+1})$
 - it is an *estimate* like MC target because it **samples** the expected value
 - it is an *estimate* like the DP target because it uses the current estimate of V instead of v_{π}

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

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

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

A action given by π for S

Take action A , observe R, S'

$V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

until S is terminal

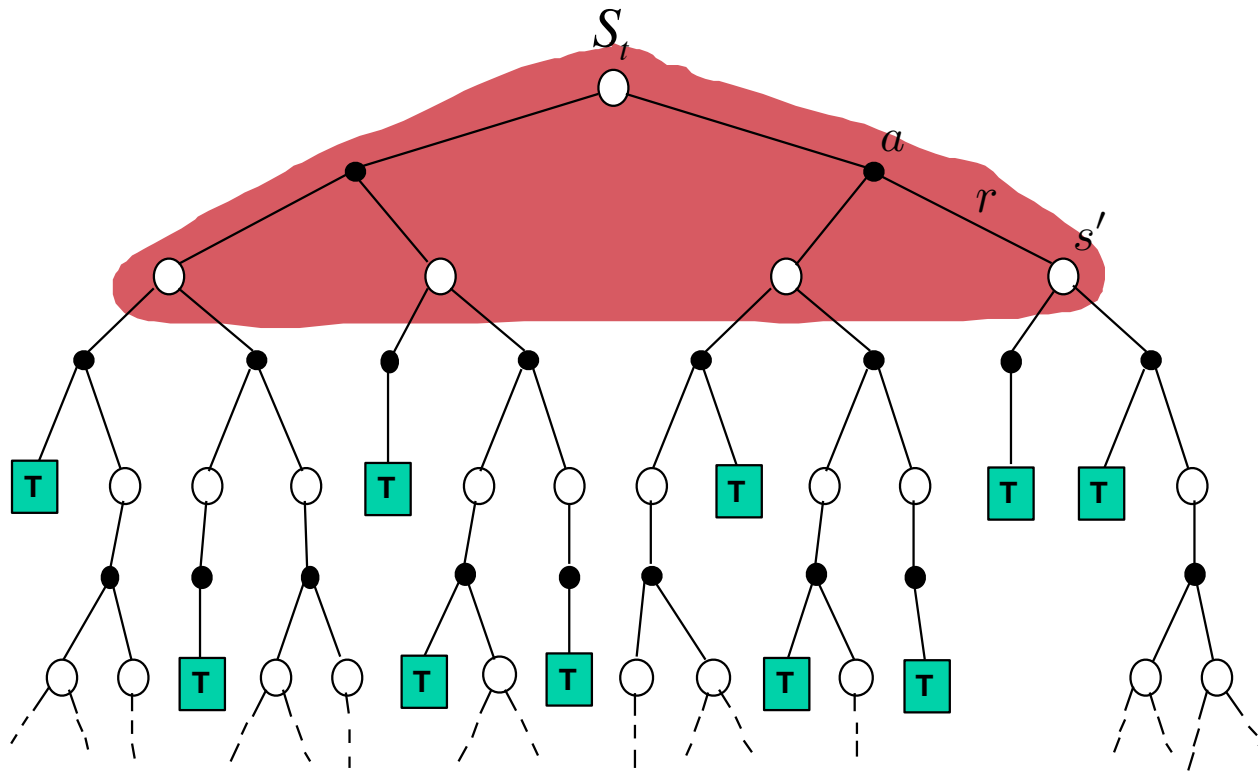
Agent program

Environment program

Experiment program

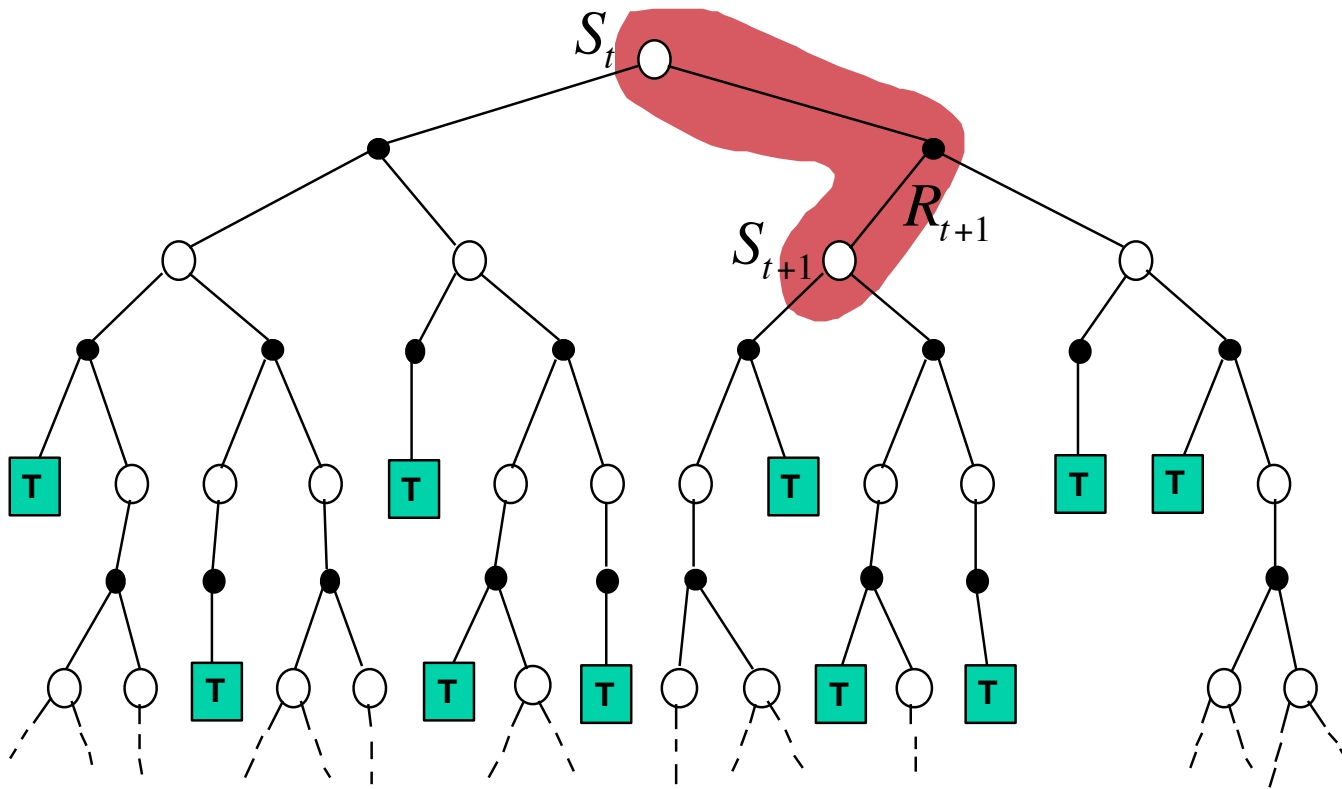
Dynamic programming

$$V(S_t) \leftarrow E_{\pi} [R_{t+1} + \gamma V(S_{t+1})] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a) [r + \gamma V(s')]$$



Simplest TD method

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$



Example: Driving Home

- ▶ Consider driving home:
 - each day you drive home
 - your goal is to try and predict how long it will take at particular stages
 - when you leave office you note the time, day, & other relevant info
- ▶ Consider the policy evaluation or prediction task

Driving Home

<i>State</i>	<i>Elapsed Time (minutes)</i>	<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Driving home as an RL problem

- ▶ Rewards = 1 per step (if we were minimizing travel time what would reward be?)
- ▶ $\gamma = 1$
- ▶ $G_t =$ time to go from state S_t
- ▶ $V(S_t) =$ expected time to get home from S_t

Updating our predictions

- ▶ Goal: update the prediction of total time leaving from office, while driving home

Driving home

<i>State</i>	<i>Elapsed Time</i> <i>(minutes)</i>	V(s)	V(office)
		<i>Predicted</i> <i>Time to Go</i>	<i>Predicted</i> <i>Total Time</i>
leaving office, friday at 6	0	5	30
reach car, raining	5	15	40
exiting highway	20	10	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

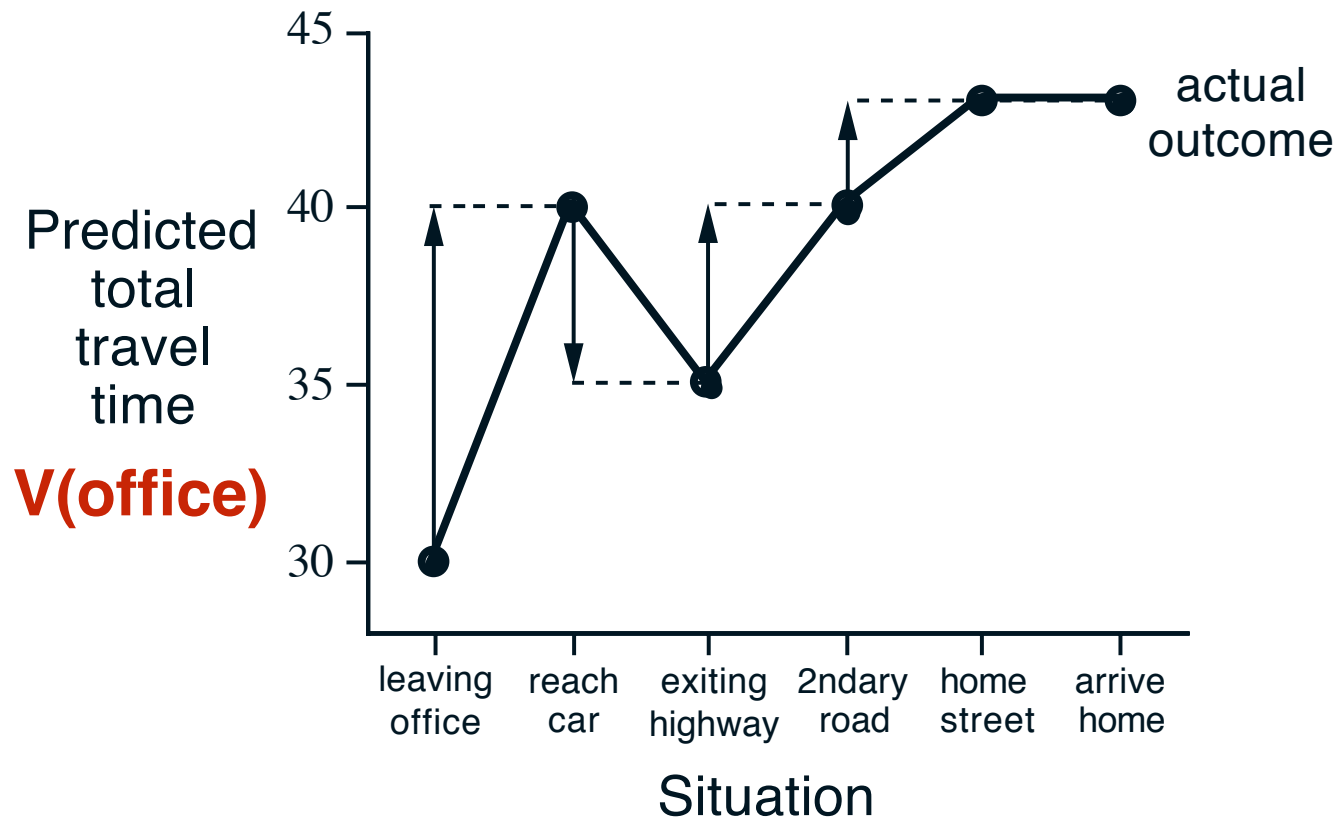
- ▶ Task: update the value function as we go, based on observed elapsed time—Reward column

Driving home

<i>State</i>	<i>Elapsed Time (minutes)</i>	R	V(s)	V(office)
			<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43	0	0	43

- ▶ update $V(\text{office})$ with $\alpha = 1$?
 - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
 - $V(\text{office}) = V(\text{office}) + \alpha[R_{t+1} + \gamma V(\text{car}) - V(\text{office})]$
 - new $V(\text{office}) = 40$; $\Delta = +10$
- ▶ update $V(\text{car})$?
 - $V(\text{car}) = 30$; $\Delta = -5$
- ▶ update $V(\text{exit})$?
 - $V(\text{exit}) = 20$; $\Delta = +5$

Changes recommended by TD methods ($\alpha = 1$)



Advantages of TD learning

- ▶ TD methods do not require a model of the environment, only experience
- ▶ TD methods can be fully incremental
 - ▶ Make updates **before** knowing the final outcome
 - ▶ Requires less memory
 - ▶ Requires less peak computation
- ▶ You can learn **without** the final outcome, from incomplete sequences

Optimality of TD(0)

- ▶ TD(0) achieves a **special** type of optimality
 - This is correct for the maximum likelihood estimate of the Markov model generating the data
 - i.e., if we do a best fit Markov model, and assume it is exactly correct, and then compute the predictions
 - This is called the **certainty-equivalence estimate**

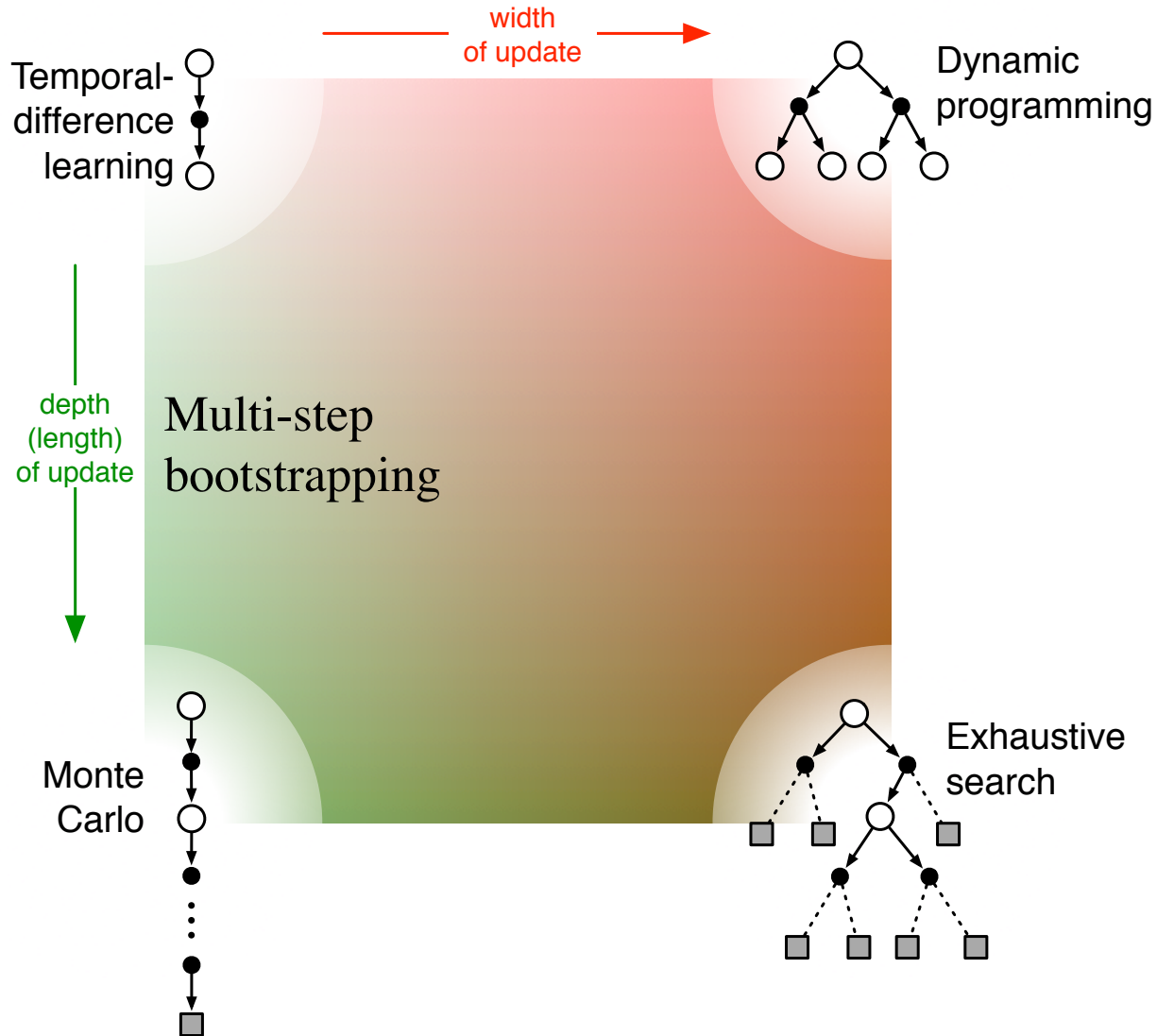
Advantages of TD

- ▶ If the process is Markov, then we expect the TD estimate to produce lower error on future data
- ▶ This helps explain why TD methods converge more quickly than MC in the batch setting
- ▶ TD(0) makes progress towards the certainty-equivalence estimate without explicitly building the model!

Summary so far

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods

Unified View



Off-policy methods

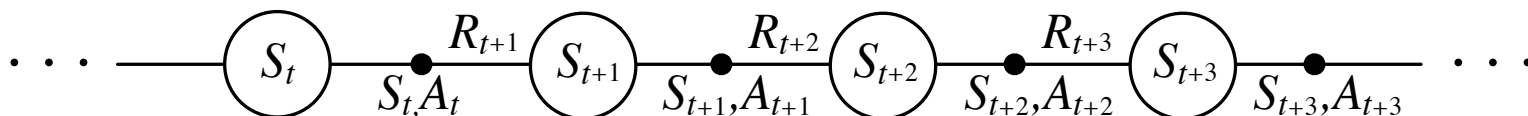
- ❑ Learn the value of the *target policy* π from experience due to *behavior policy* b
- ❑ For example, π is the greedy policy (and ultimately the optimal policy) while μ is exploratory (e.g., ϵ -soft)
- ❑ In general, we only require *coverage*, i.e., that b generates behavior that covers, or includes, π

$$\pi(a|s) > 0 \quad \text{for every } s, a \text{ at which } b(a|s) > 0$$

- ❑ Idea: *importance sampling*
 - Weight each return by the *ratio of the probabilities* of the trajectory under the two policies

Learning An Action-Value Function

Estimate q_π for the current policy π



After every transition from a nonterminal state, S_t , do this:

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

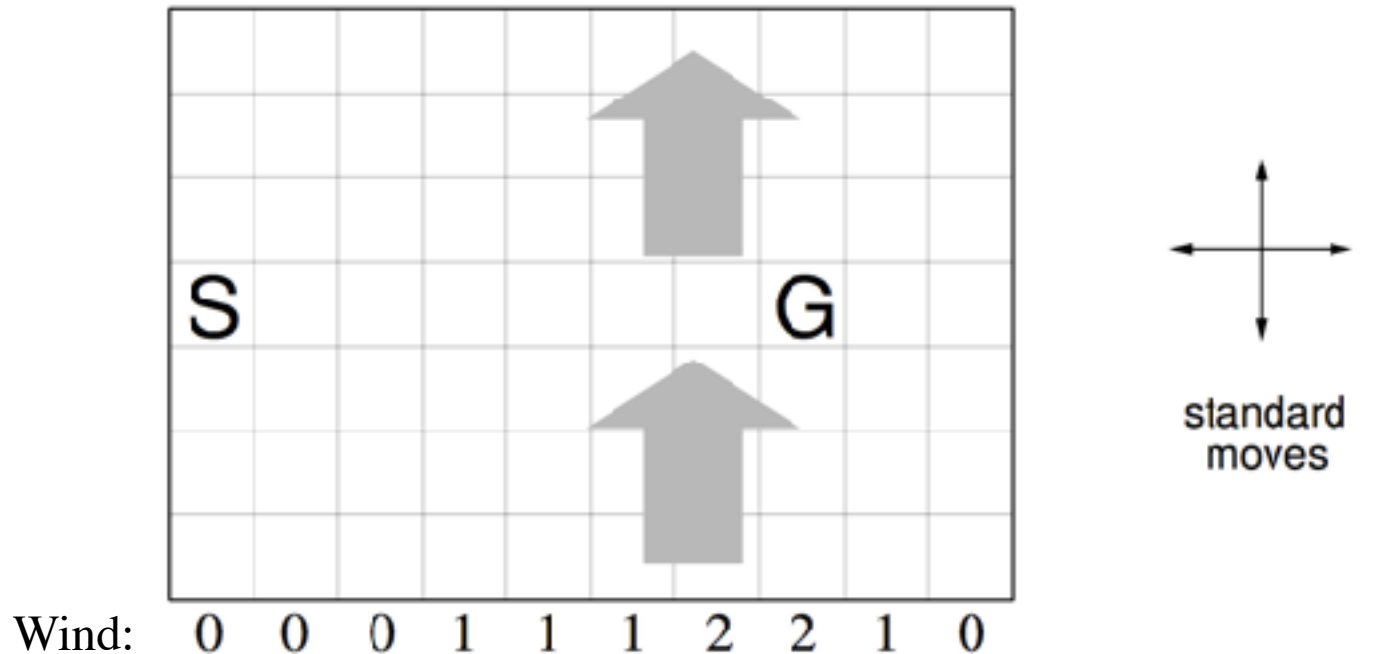
If S_{t+1} is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

Sarsa: On-Policy TD Control

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

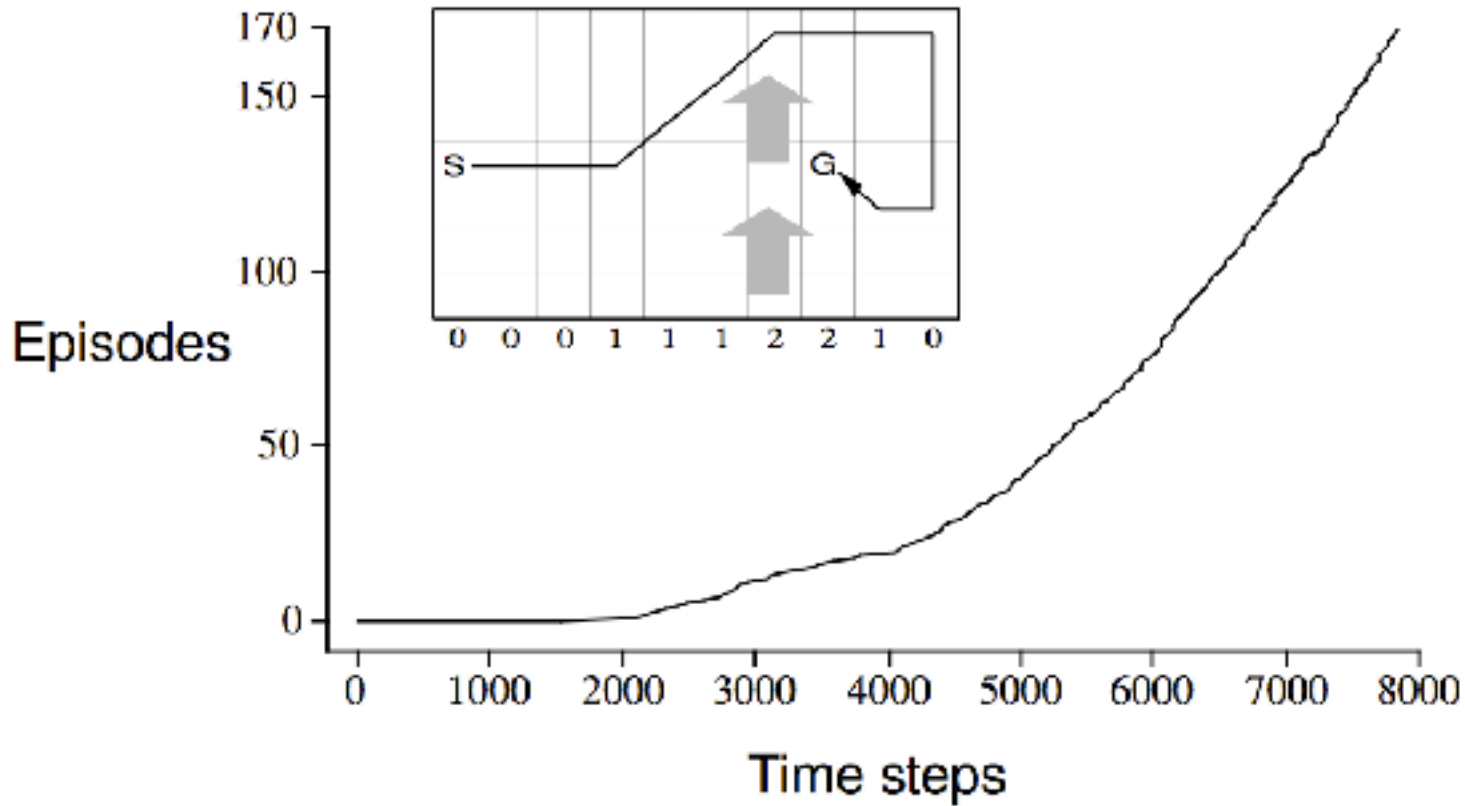
```
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A'$ 
  until  $S$  is terminal
```

Windy Gridworld



undiscounted, episodic, reward = -1 until goal

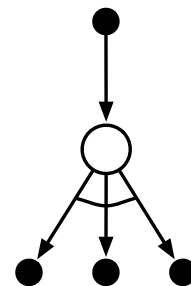
Results of Sarsa on the Windy Gridworld



Q-Learning: Off-Policy TD Control

One-step Q-learning:

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



Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

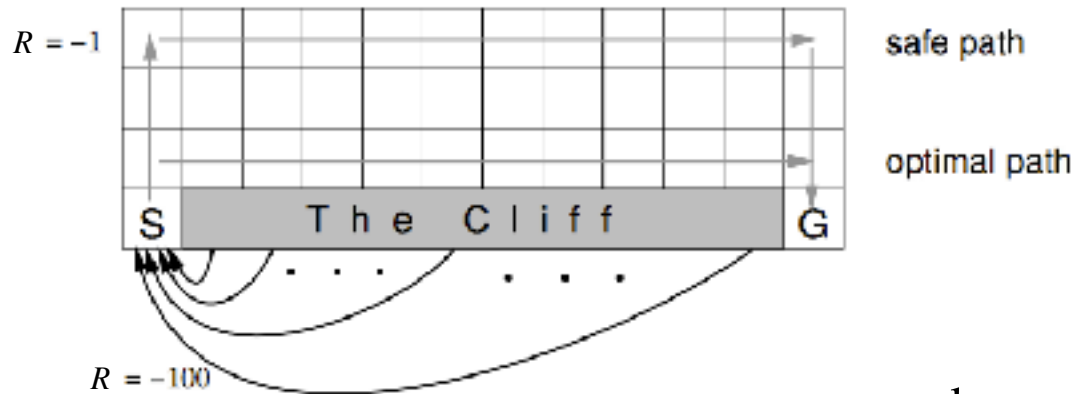
Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

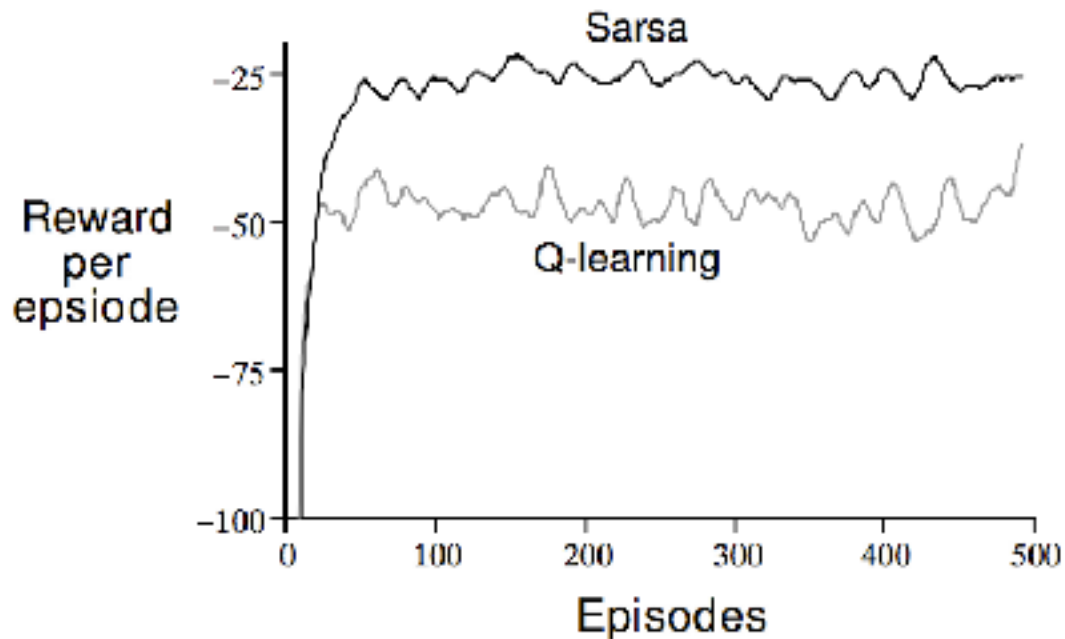
$S \leftarrow S'$;

until S is terminal

Cliffwalking



ϵ -greedy, $\epsilon = 0.1$



Summary

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data
- Extend prediction to control by employing some form of GPI
 - On-policy control: *Sarsa, Expected Sarsa*
 - Off-policy control: *Q-learning, Expected Sarsa*
- Avoiding maximization bias with Double Q-learning