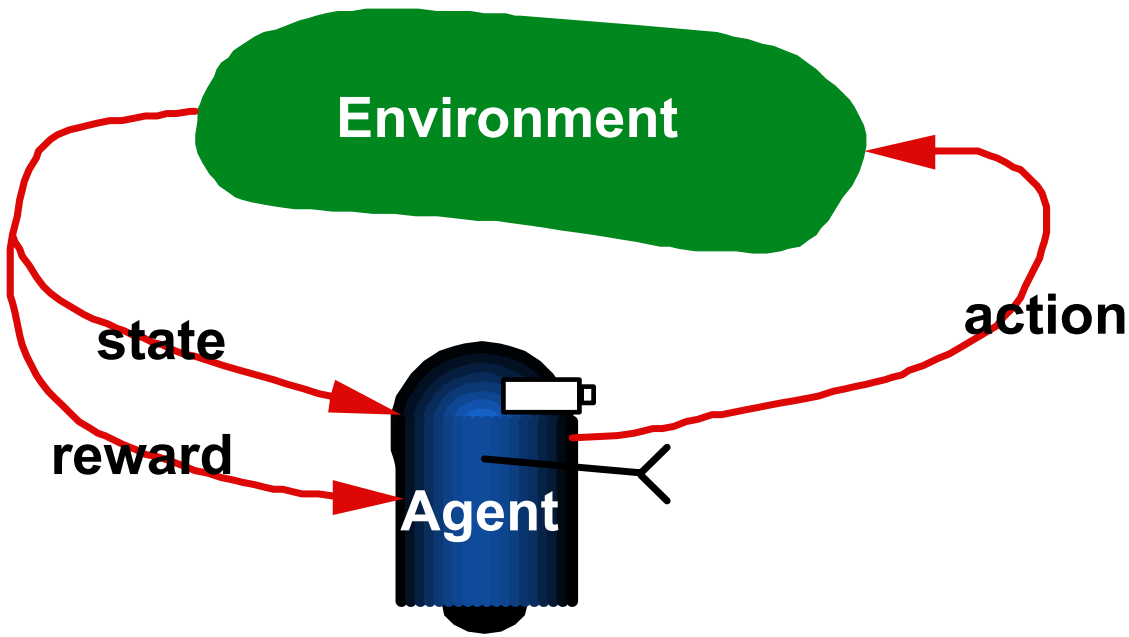
What is Reinforcement Learning?

- An approach to Artificial Intelligence
- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal

Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to ***affect*** the environment
- Environment is stochastic and uncertain



adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

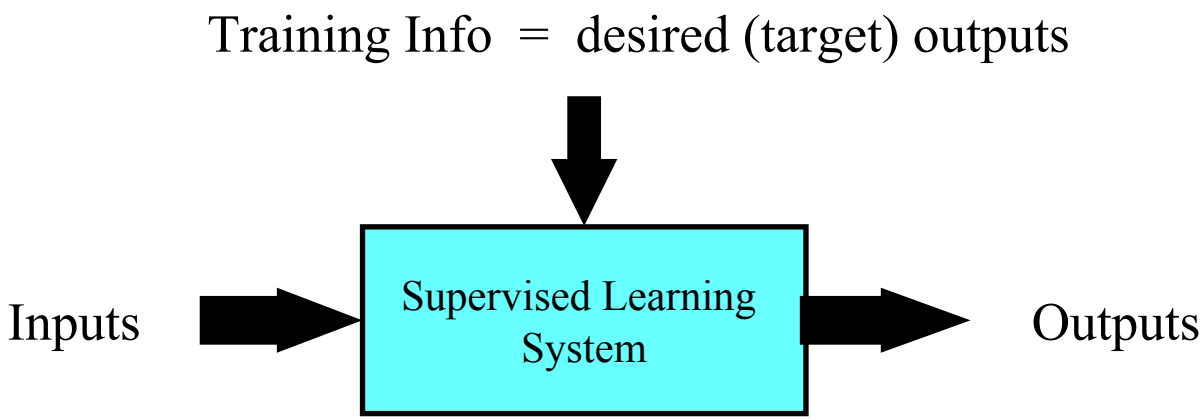
Key Features of RL

- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to ***explore*** and ***exploit***
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Examples of Reinforcement Learning

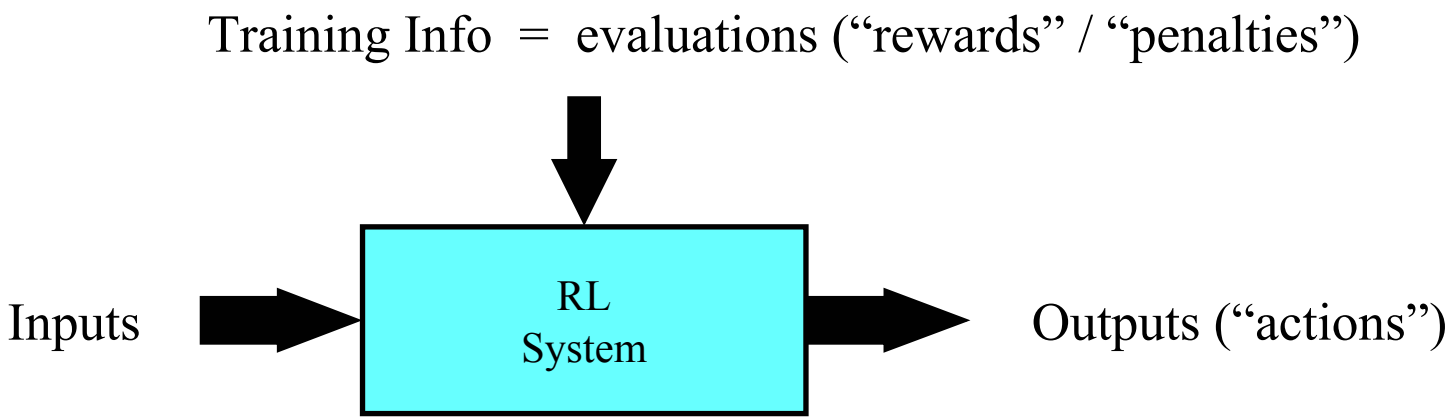
- Robocup Soccer Teams Stone & Veloso, Riedmiller et al.
  - World's best player of simulated soccer, 1999; Runner-up 2000
- Inventory Management Van Roy, Bertsekas, Lee & Tsitsiklis
  - 10-15% improvement over industry standard methods
- Dynamic Channel Assignment Singh & Bertsekas, Nie & Haykin
  - World's best assigner of radio channels to mobile telephone calls
- Elevator Control Crites & Barto
  - (Probably) world's best down-peak elevator controller
- Many Robots
  - navigation, bi-pedal walking, grasping, switching between skills...
- TD-Gammon and Jellyfish Tesauro, Dahl
  - World's best backgammon player

Supervised Learning



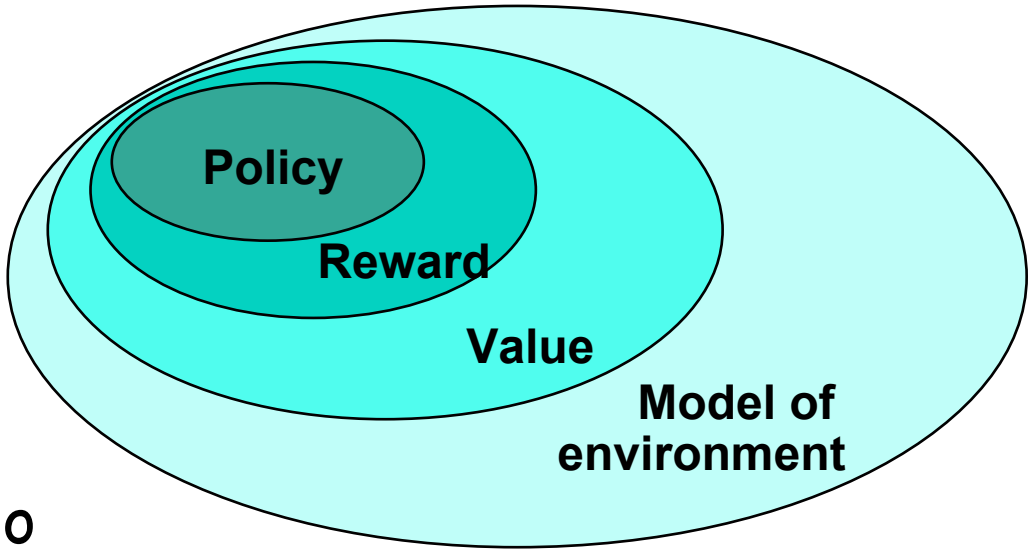
Error = (target output – actual output)

Reinforcement Learning



Objective: get as much reward as possible

Elements of RL



- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what

ε-Greedy Action Selection

- Greedy action selection:

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- ε-Greedy:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\varepsilon \\ \text{random action with probability } \varepsilon \end{cases}$$

. . . the simplest way to balance exploration and exploitation

Softmax Action Selection

- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

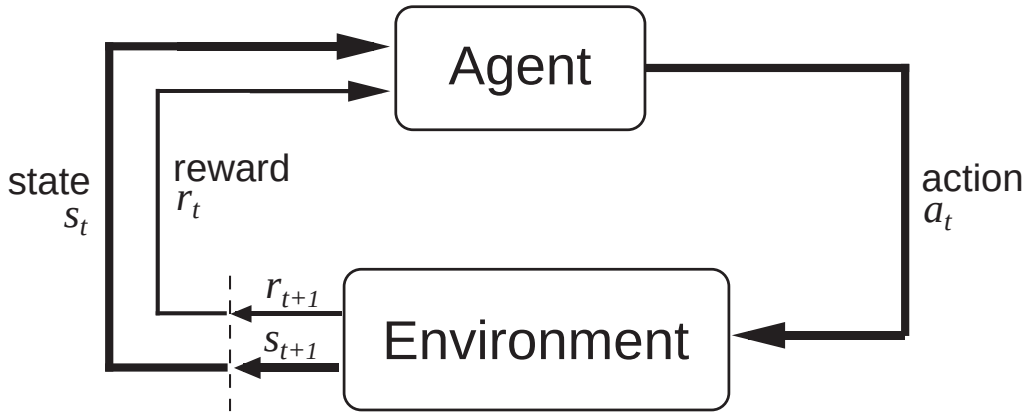
Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

where τ is the

"computational temperature"

The Agent-Environment Interface



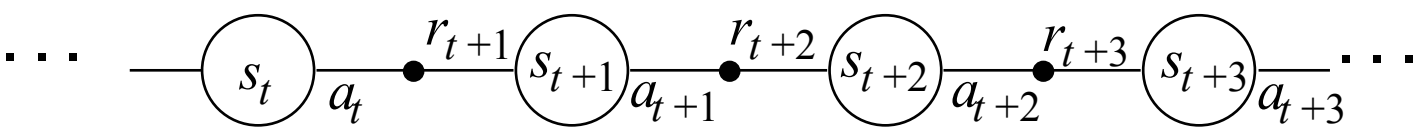
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

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}



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.

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 :

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

What do we want to maximize?

In general,

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

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

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

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.

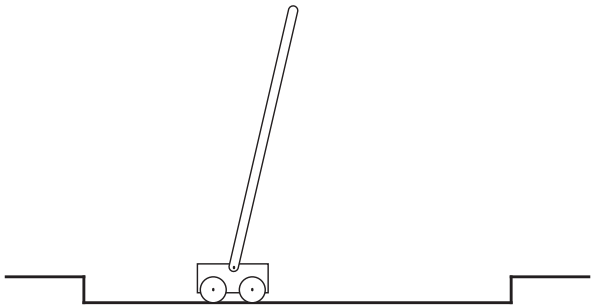
Discounted return:

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

where $\gamma, 0 \le \gamma \le 1,$ 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

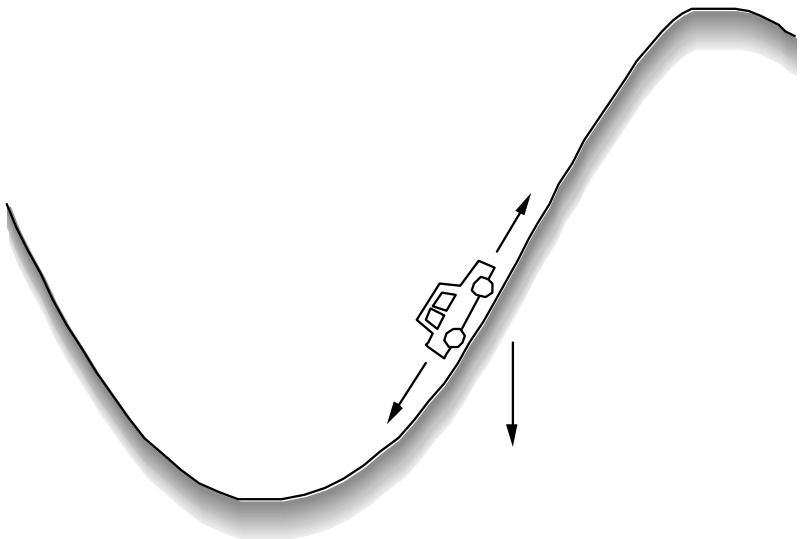
As a **continuing task** with discounted return:

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.

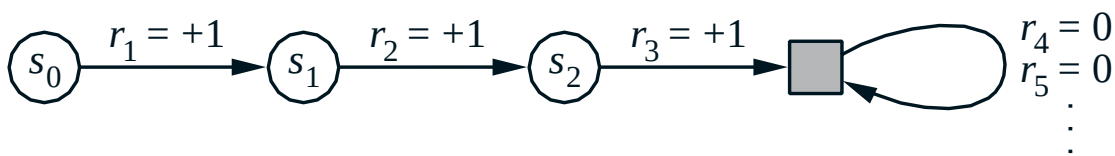
reward $= -1$ for each step where **not** at top of hill

$\Rightarrow$ 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

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

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

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**:

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

$$\Pr\left\{s_{t+1}=s', r_{t+1}=r \,\middle|\, s_t, a_t\right\}$$

for all s', r, and 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**:

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

  - **reward probabilities**:

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

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 runs out of power while searching, 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

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

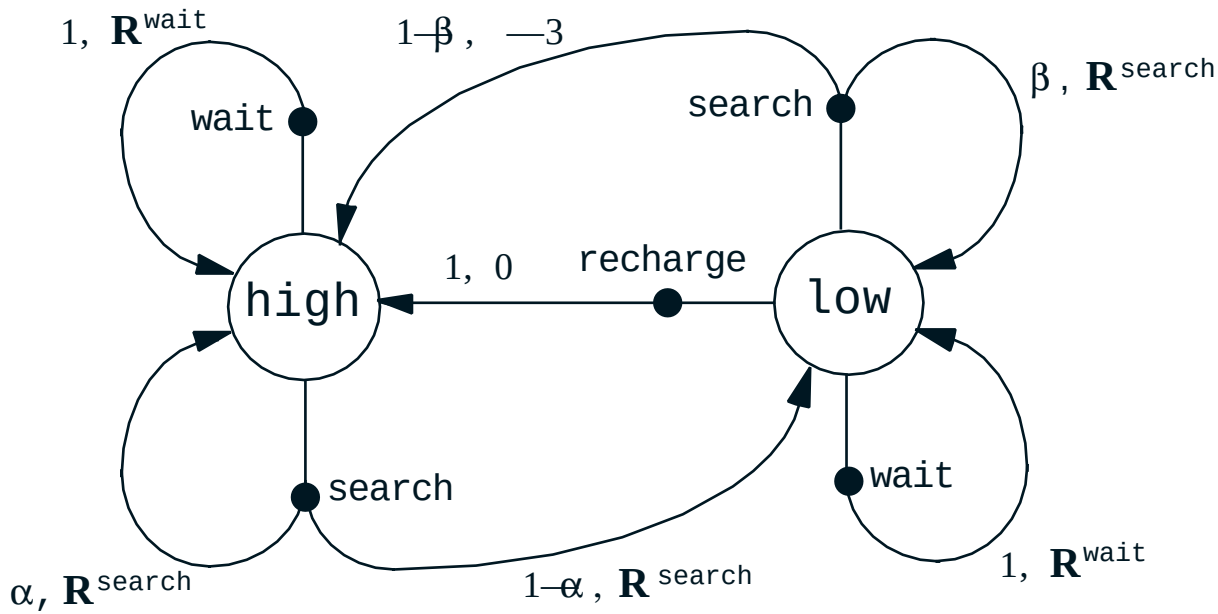
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

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

$\mathbf{R}^{\texttt{search}}$ = 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:

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

- 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:

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

Bellman Equation for a Policy π

The basic idea:

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

So:

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

Or, without the expectation operator:

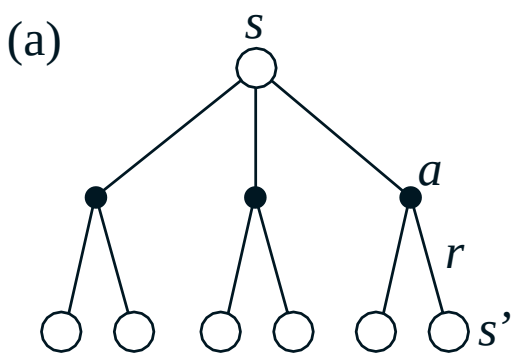
$$
V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^{\pi}(s') \right]
$$

More on the Bellman Equation

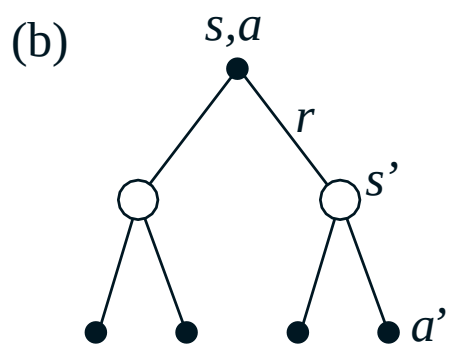
$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathbf{P}^{a}_{ss'} \left[\mathbf{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

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

Backup diagrams:



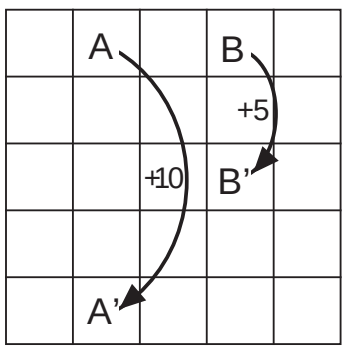
for V^{π}



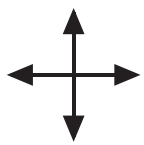
for Q^{π}

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; $\gamma = 0.9$

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**:

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

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

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

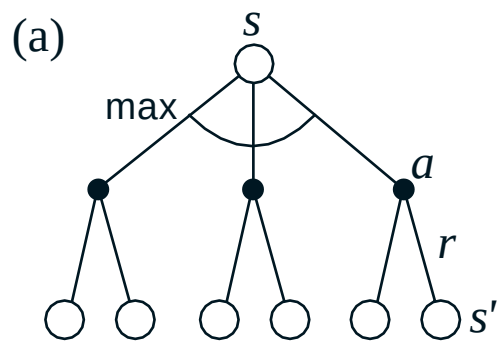
This is the expected return for taking action a in state s and thereafter following an optimal policy.

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:



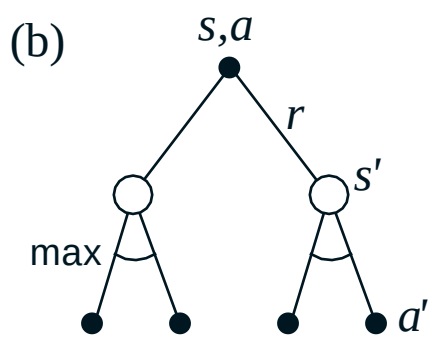
V^* is the unique solution of this system of nonlinear equations.

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:



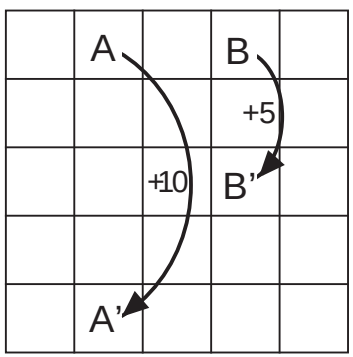
Q^* is the unique solution of this system of nonlinear equations.

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.

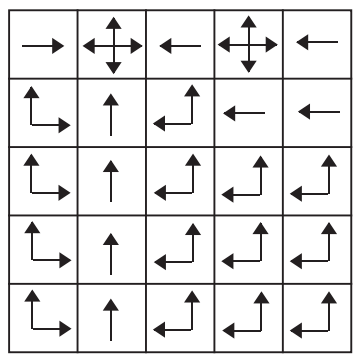
E.g., back to the gridworld:



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^*

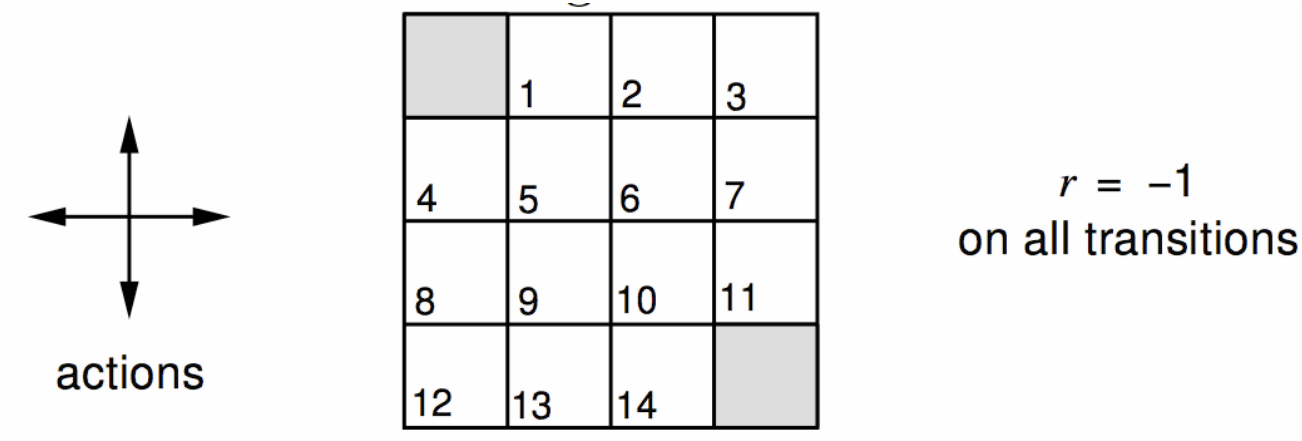


c) π^*

Iterative Policy Evaluation

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

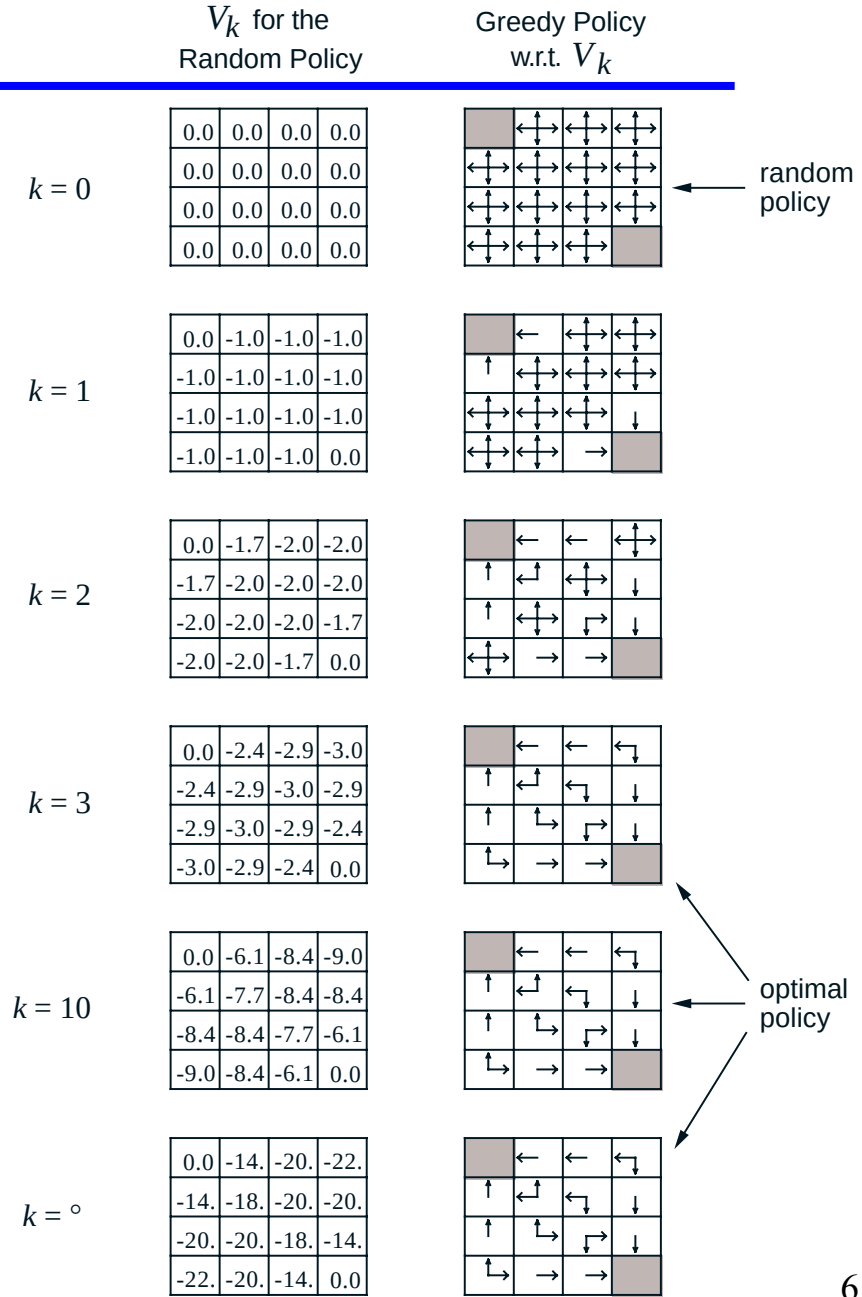
A Small Gridworld



- 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

$\pi =$ equiprobable random action choices



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 $a \neq \pi(s)$?

The value of doing a in state s is :

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

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

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

$$Q^\pi(s,a) > V^\pi(s)$$

Policy Improvement Cont.

Do this for all states to get a new policy π' that is **greedy** with respect to V^{π} :

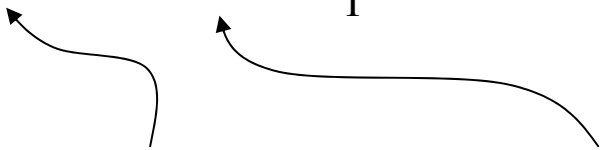
$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

$$= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right]$$

Then $V^{\pi'} \geq V^{\pi}$

Policy Iteration

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \cdots \pi^* \rightarrow V^* \rightarrow \pi^*$$



policy evaluation

policy improvement
"greedification"

Policy Iteration

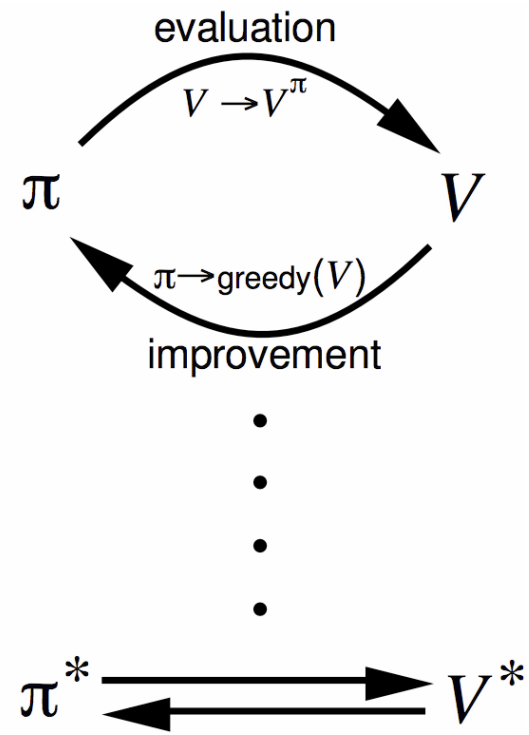
1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

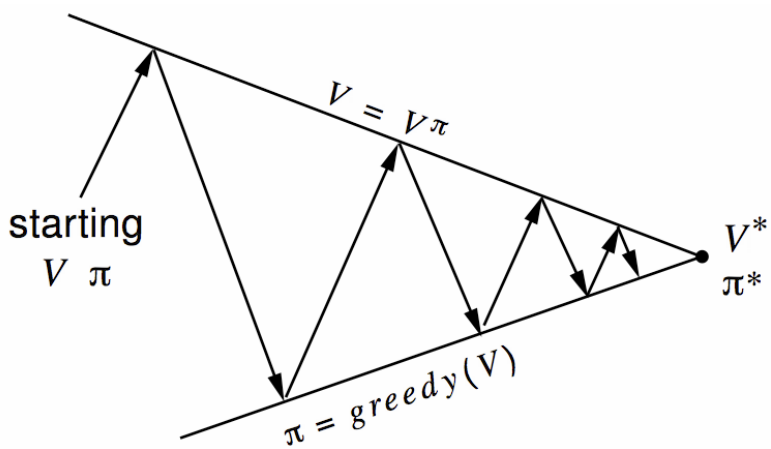
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $b \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
      If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

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:



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 Prediction

Policy Evaluation (the prediction problem):
for a given policy π, compute the state-value function V^{π}

Recall: Simple every-visit Monte Carlo method:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right]$$

target: the actual return after time t

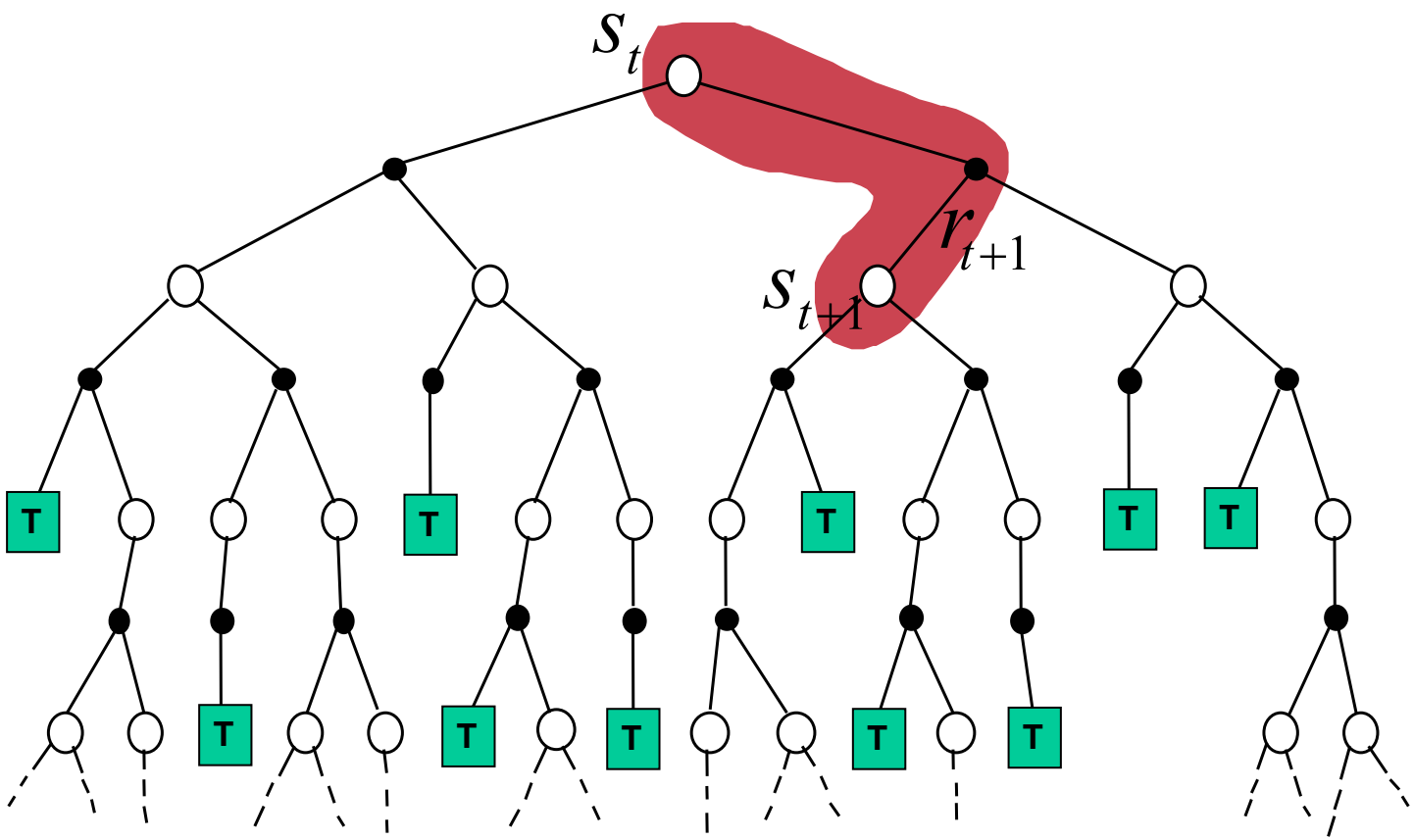
The simplest TD method, TD(0):

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

target: an estimate of the return

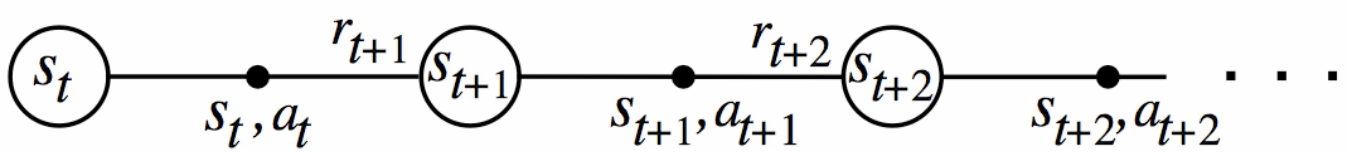
Simplest TD Method

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



Learning An Action-Value Function

Estimate Q^π for the current behavior policy π.



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

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

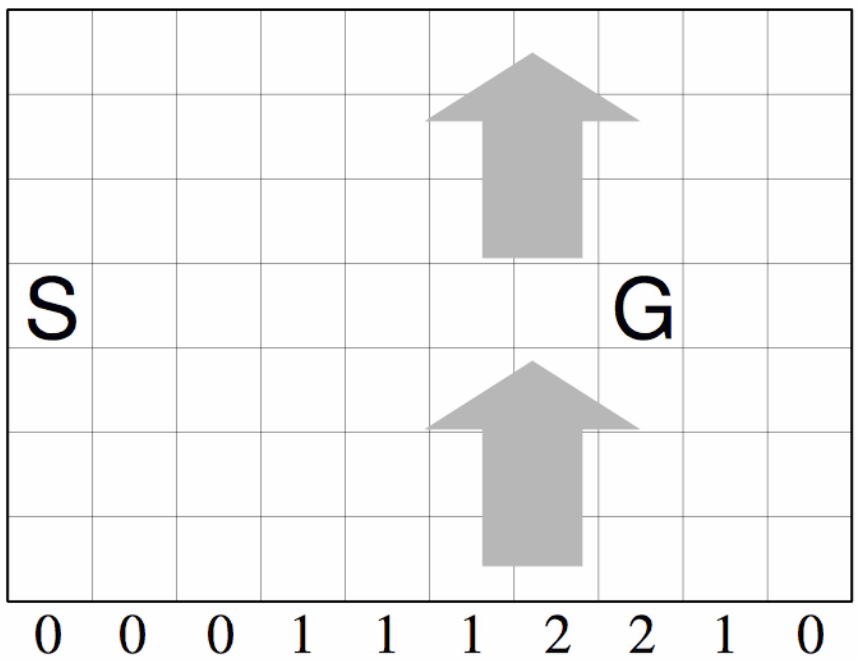
If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

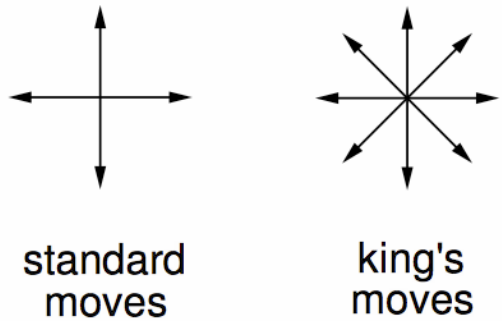
Sarsa: On-Policy TD Control

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

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

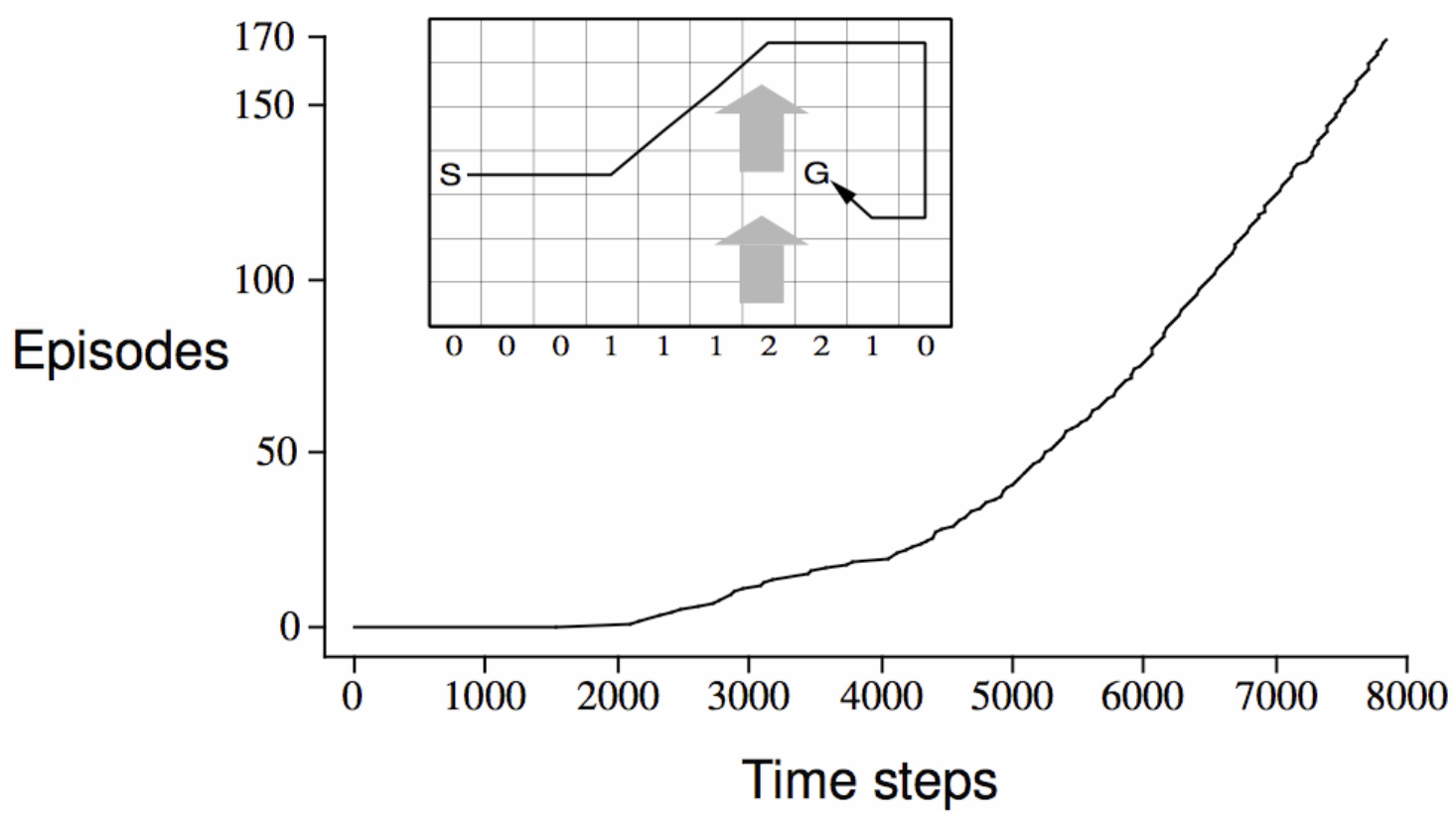
Windy Gridworld





undiscounted, episodic, reward = –1 until goal

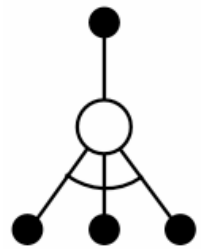
Results of Sarsa on the Windy Gridworld



Q-Learning: Off-Policy TD Control

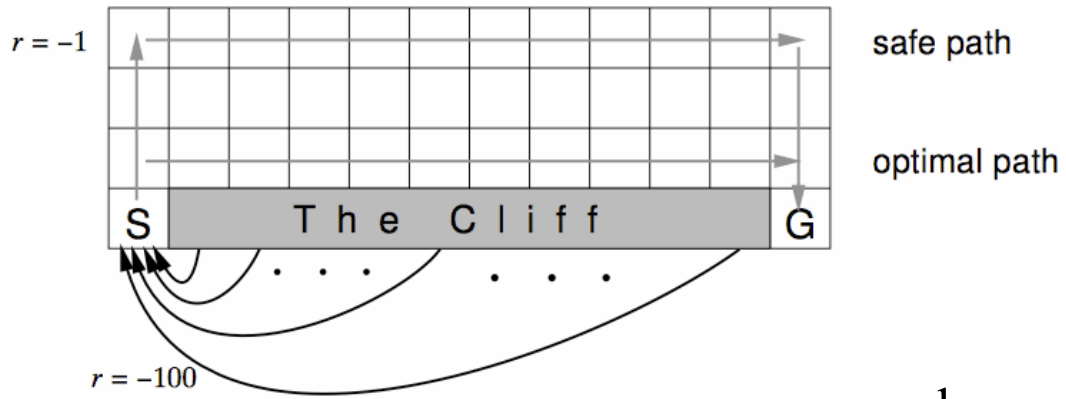
One - step Q - learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$



Initialize $Q(s, a)$ arbitrarily
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, ε = 0.1