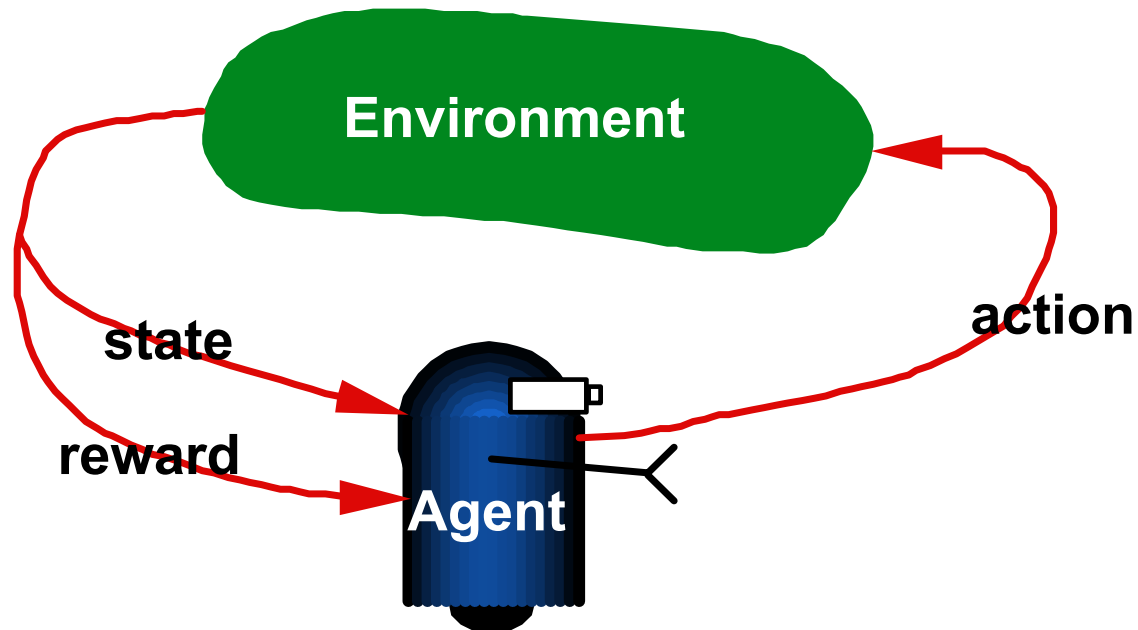


What is Reinforcement Learning?

- 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

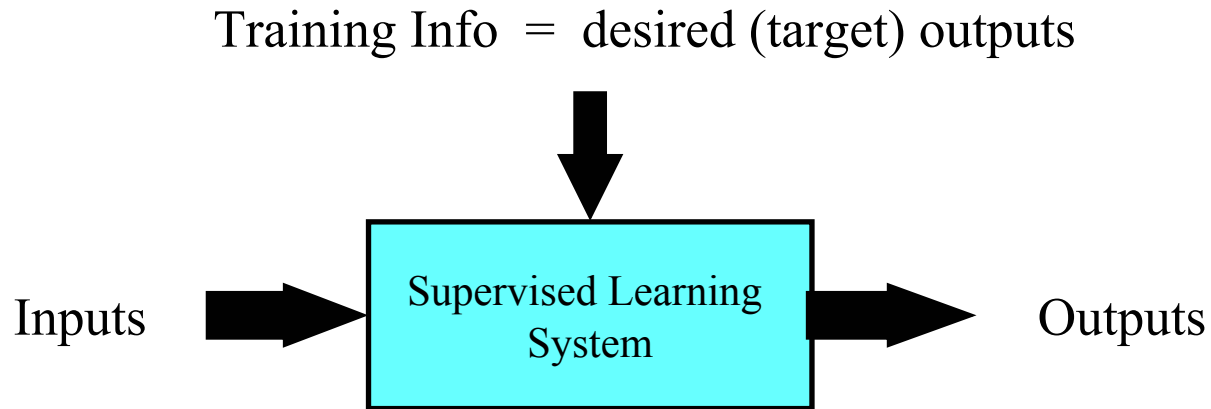
- Temporally situated
- Continual learning and planning
- Object is to *affect* the environment
- Environment is stochastic and uncertain



Key Features of RL

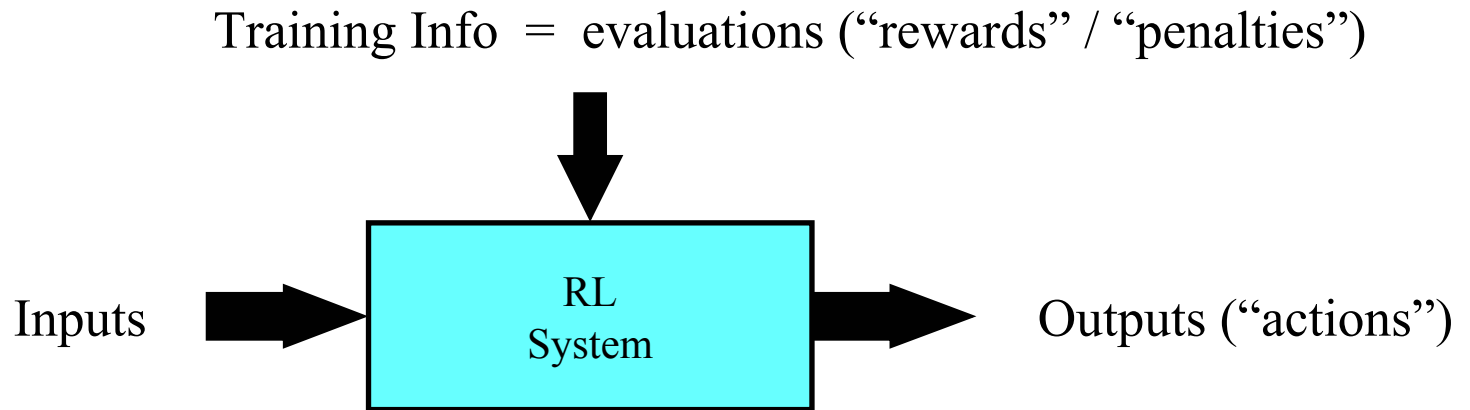
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
 - Sacrifice short-term gains for greater long-term gains
- The need to *explore* and *exploit*
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Supervised Learning



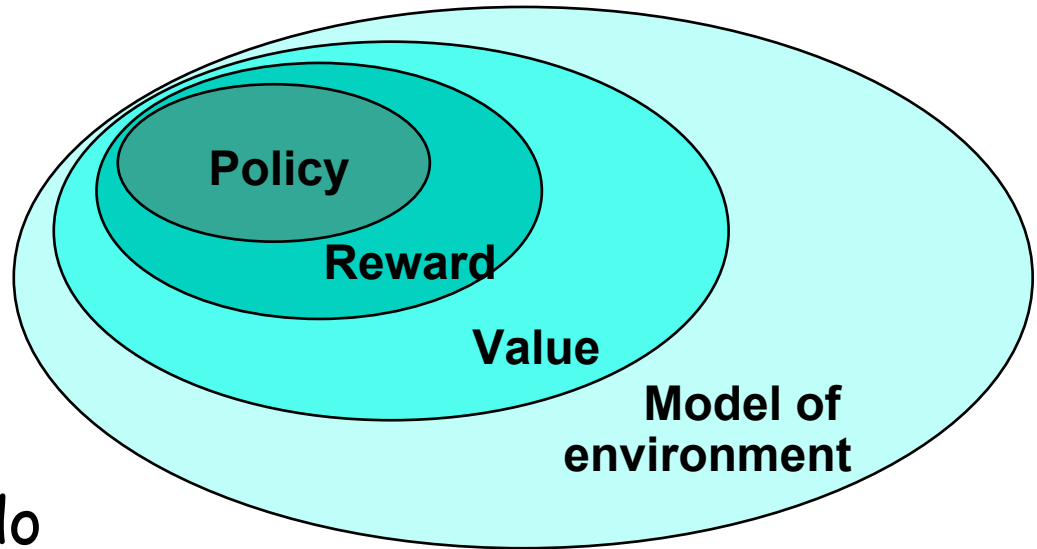
$$\text{Error} = (\text{target output} - \text{actual output})$$

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

The Exploration/Exploitation Dilemma

- ❑ Suppose you form estimates

$$Q_t(a) \approx Q^*(a) \quad \text{action value estimates}$$

- ❑ The **greedy** action at t is a_t

$$a_t^* = \operatorname{argmax}_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- ❑ You can't exploit all the time; you can't explore all the time
- ❑ You can never stop exploring; but you should always reduce exploring. Maybe.

ε -Greedy Action Selection

□ Greedy action selection:

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

□ ε -Greedy:

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

. . . 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:

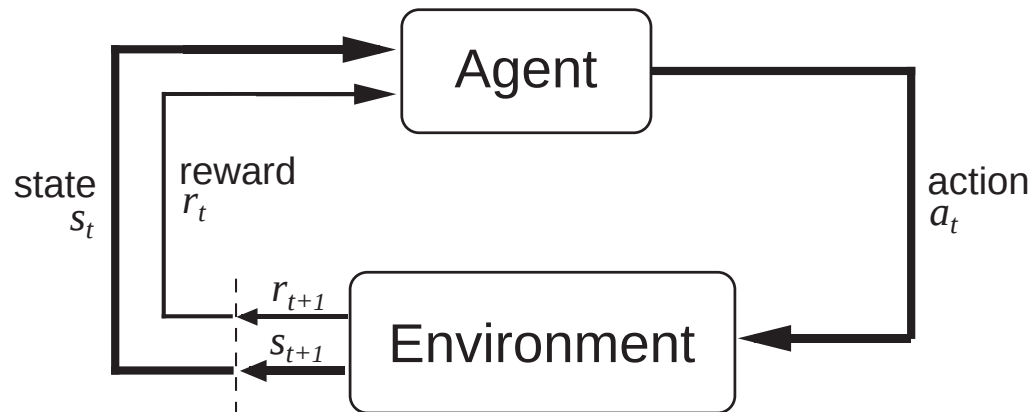
Choose action a on play t with probability

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

where τ is the

“computational temperature”

The Agent-Environment Interface



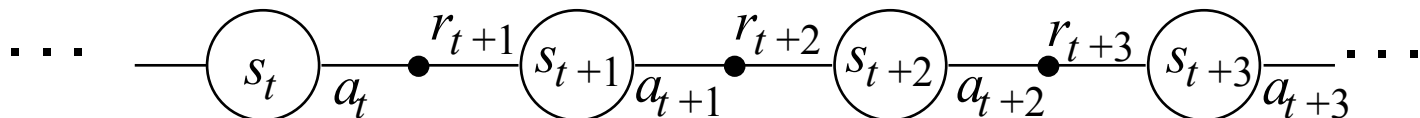
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \dots$

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

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

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

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 :

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

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.

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

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.

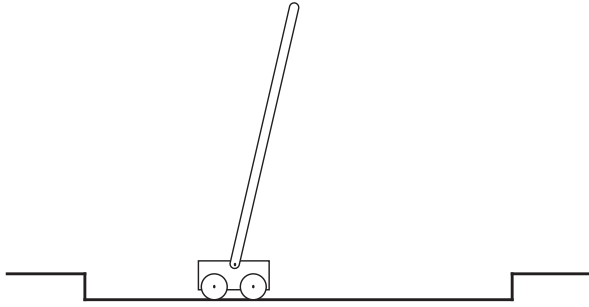
Discounted return:

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

where γ , $0 \leq \gamma \leq 1$, 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:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

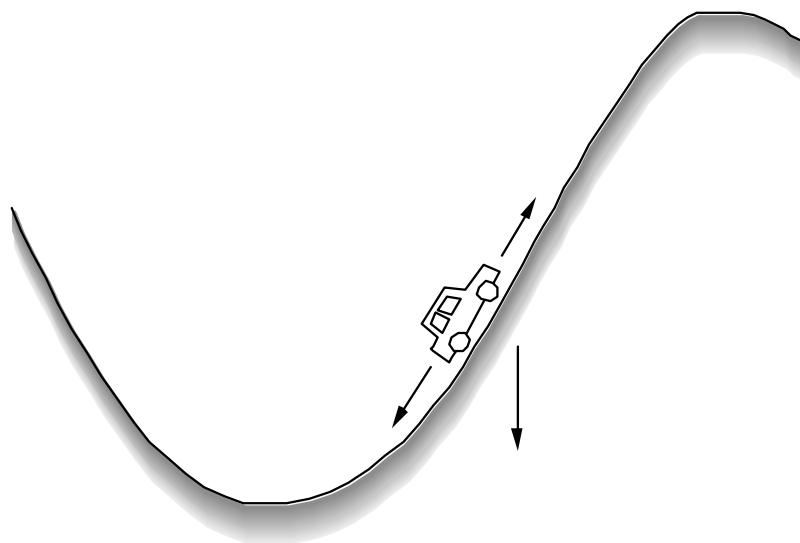
As a **continuing task** with discounted return:

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.

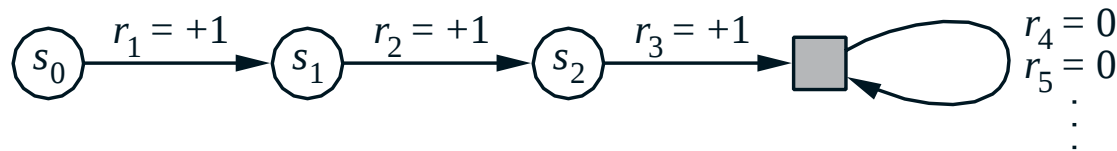
reward = -1 for each step where **not** at top of hill

$\Rightarrow$ 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 all 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**:

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

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

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**:

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

- **reward probabilities**:

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

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

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

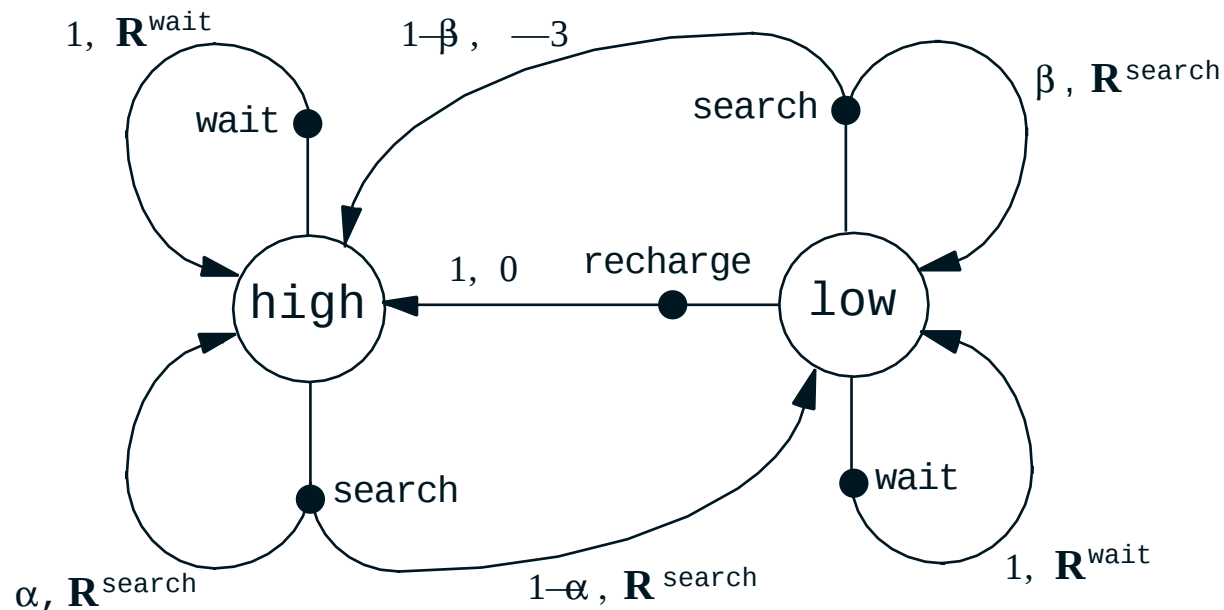
$A(\text{high}) = \{\text{search}, \text{wait}\}$

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

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

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

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



Bellman Equation for a Policy π

The basic idea:

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

So:

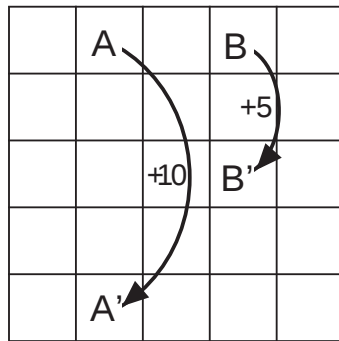
$$\begin{aligned} V^\pi(s) &= E_\pi \{R_t \mid s_t = s\} \\ &= E_\pi \{r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s\} \end{aligned}$$

Or, without the expectation operator:

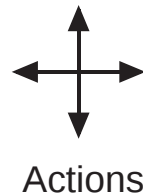
$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathbf{P}_{ss'}^a [\mathbf{R}_{ss'}^a + \gamma V^\pi(s')]$$

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)

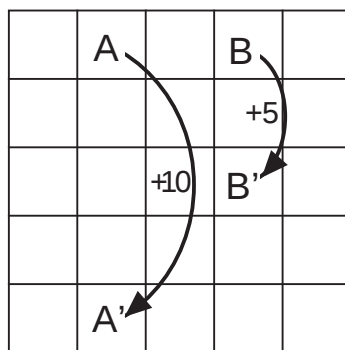
State-value function
for equiprobable
random policy;
 $\gamma = 0.9$

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.

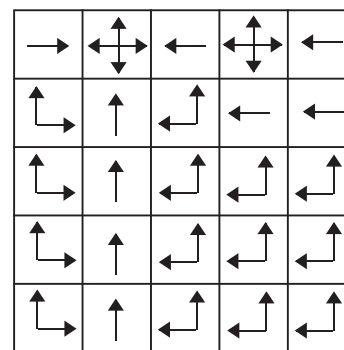
E.g., back to the gridworld:



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^*

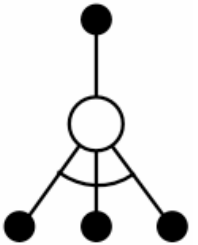


c) π^*

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