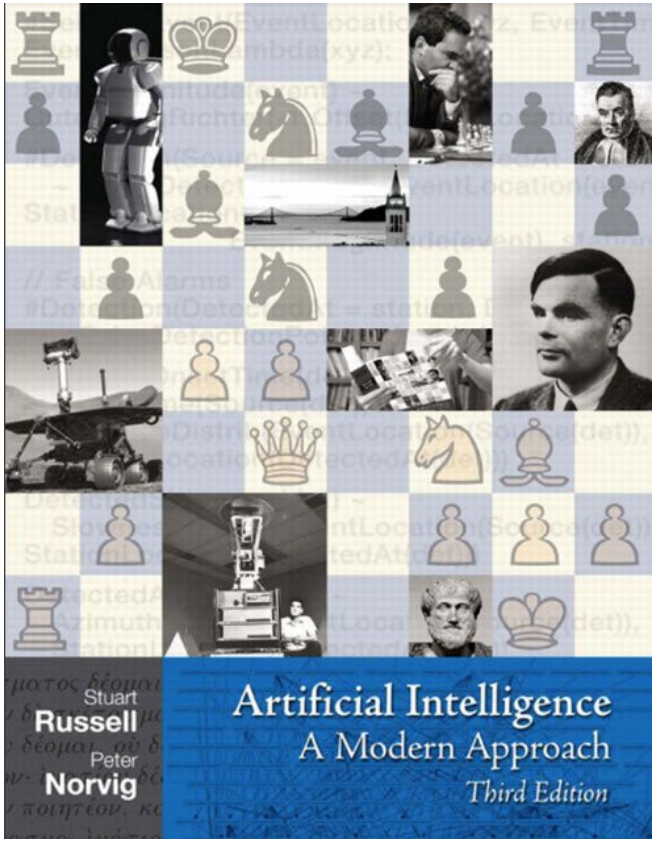
Reinforcement Learning

Chao Lan

Reference

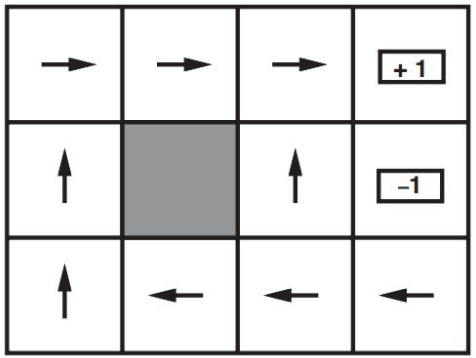
Section 21.1-21.3



Revisit: Markov Decision Process Problem

Assume reward and transition probability are known, identify optimal policy and utility.

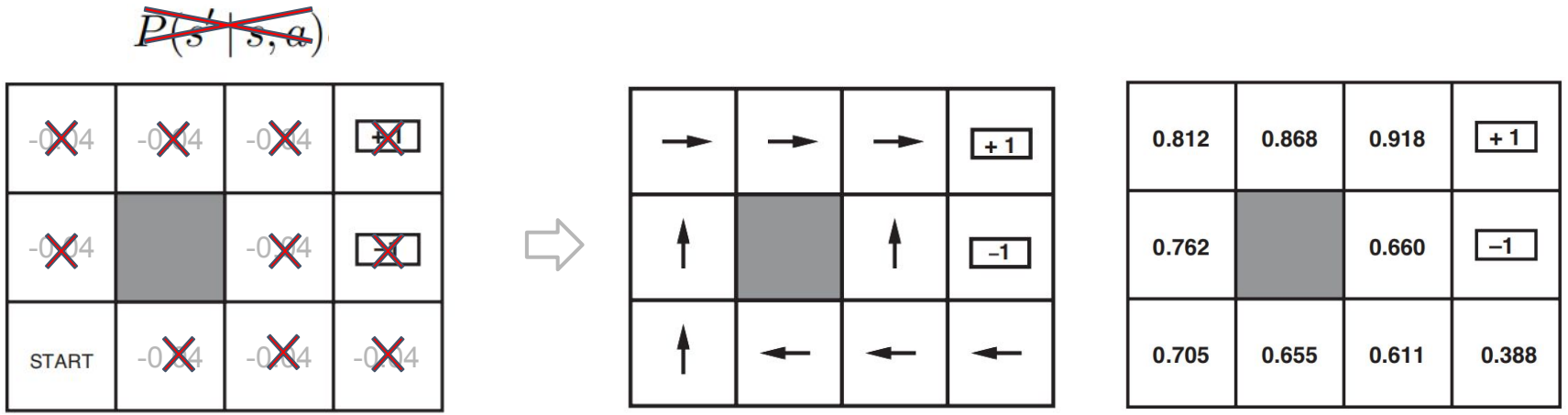
$P(s' \mid s, a)$



0.812	0.868	0.918	+1
0.762		0.660	−1
0.705	0.655	0.611	0.388

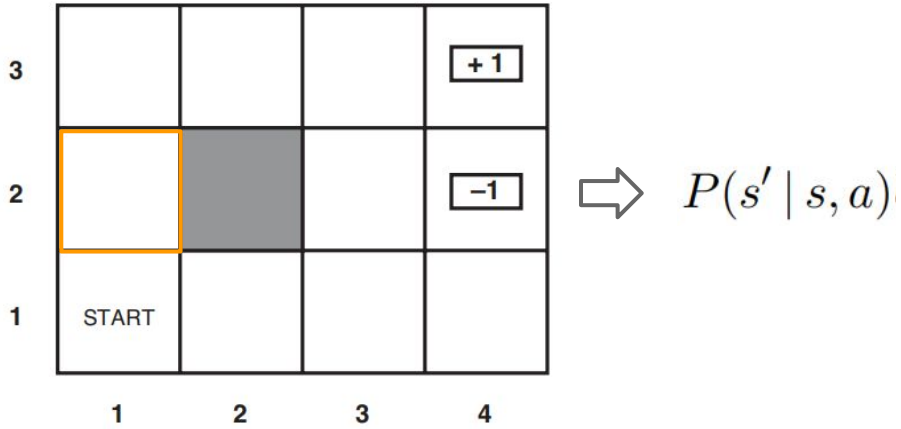
Reinforcement Learning Problem

Assume reward and transition probability are unknown, identify optimal policy and/or utility.



We also assume the unknowns are fixed and can be explored by an agent.

An Example of Exploration



Ask an agent to take action Up at (1,2) for 10 times. If it goes up 7 times, then

P(s' = (1,3) | s = (1,2), a = up) = 0.7

If it goes down 1 time, then

P(s' = (1,3) | s = (1,2), a = up) = 0.7

……

* The above is not any formal algorithm, only an example of possible exploration.

Two Types of Reinforcement Learning Problem

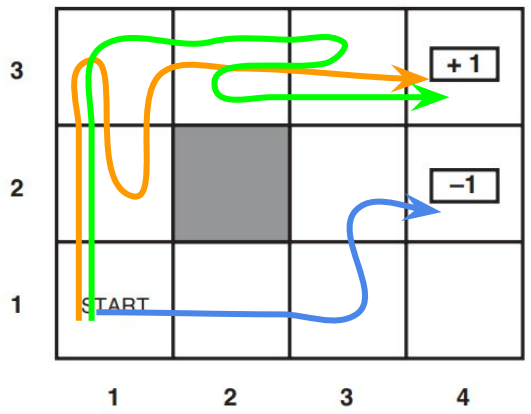
Passive RL: Given an arbitrary policy, how to identify state utility (without knowing reward and transition probability upfront)?

- Solution 1: Policy Iteration + Monte Carlo (same as in MDP)
- Solution 2: Temporal Difference Learning

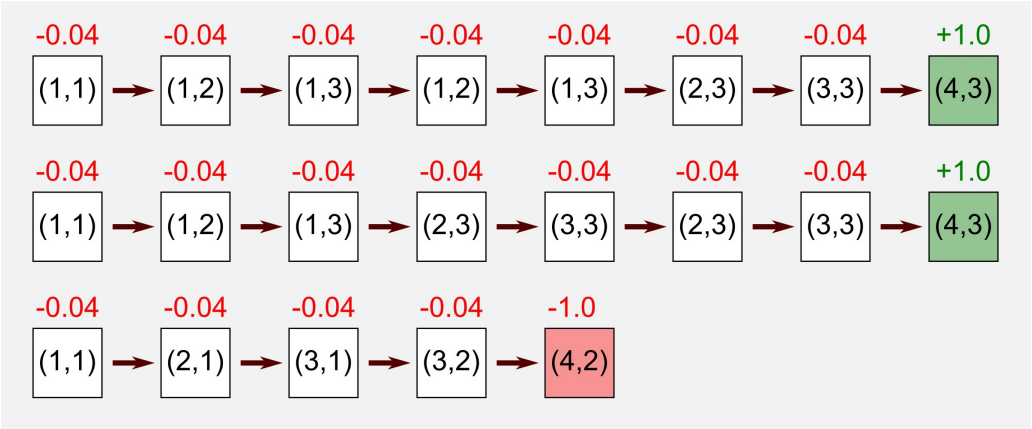
Active RL: How to identify the optimal policy (without knowing reward and transition probability upfront)?

- Solution: Q-Learning

Passive RL Solution 1: Policy Iteration + Monte Carlo



$$U(1,1) \approx (0.72+0.72-1.16) / 3 = 0.093$$



Although the agent does not know transition probability, he still moves according to it. So we can estimate utility (based on a policy) without explicitly estimating the transition probability.

Passive RL Solution 2: Temporal Difference Learning

0. Randomly initialize utilities.

1. Put agent at target state s = s_0 and set learning rate α.

2. Agent takes action π(s) and moves to s'.

3. If s' is newly visited, set R(s')=r and U(s')=r.

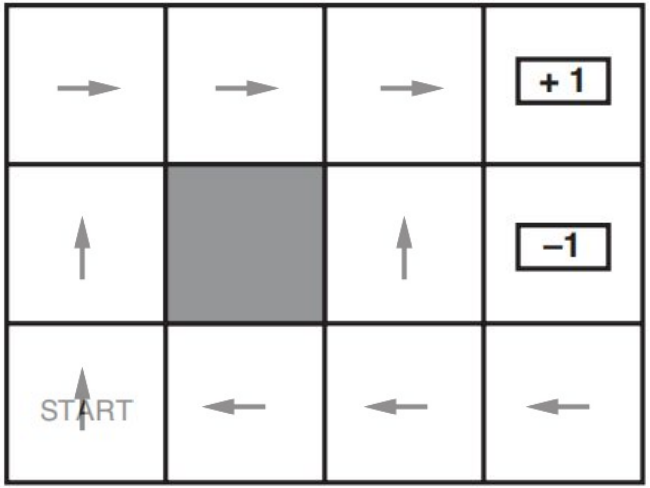
4. Update utility of s by $U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$

5. Update s=s' and repeat 2-5 until convergence. (If agent terminates, put it back to s_0.)

Basic idea is to update the utility (and reward) of a state right *after* visiting it.

Example

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$



U(1,1)	U(2,1)	U(1,2)
0	0	0

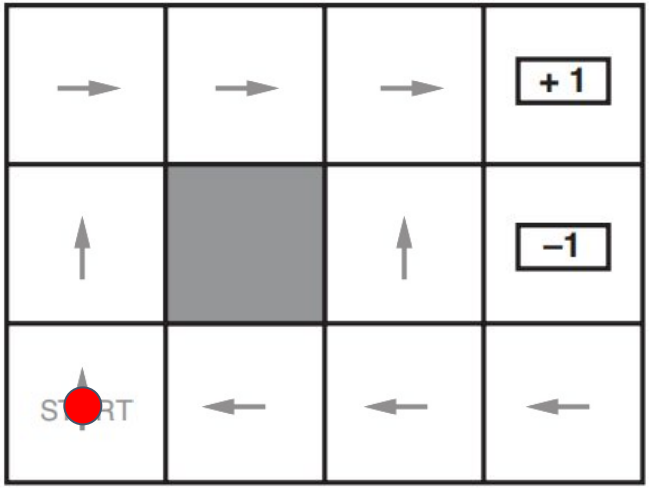
Initialization

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

1. place agent at initial state (1,1).

2. choose a learning rate, e.g., α = 0.6.

3. newly visit (1,1), set R(1,1)=r and U(1,1)=r.



U(1,1)	U(2,1)	U(1,2)
r	0	0

Iteration 1

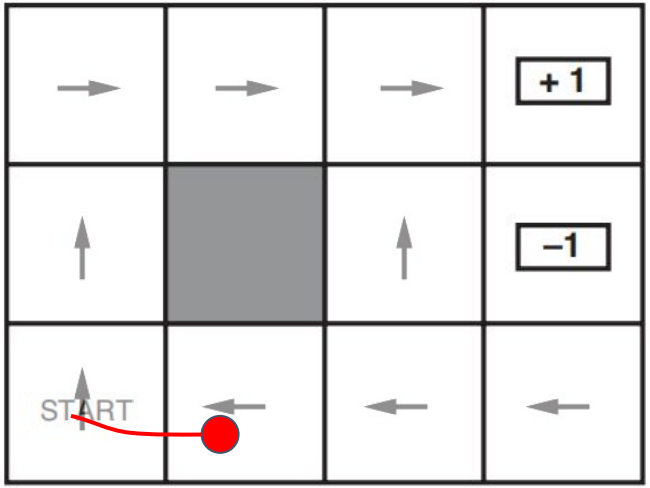
$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

1. take action UP at (1,1), end up moving right.

2. newly visit (2,1), set R(2,1) = r and U(2,1) = r

3. update utility at (1,1)

U(1,1) □ U(1,1) + α { R(1,1) + γ U(2,1) - U(1,1) }



U(1,1)	U(2,1)	U(1,2)
r + α*γ*r	r	0

Iteration 2

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

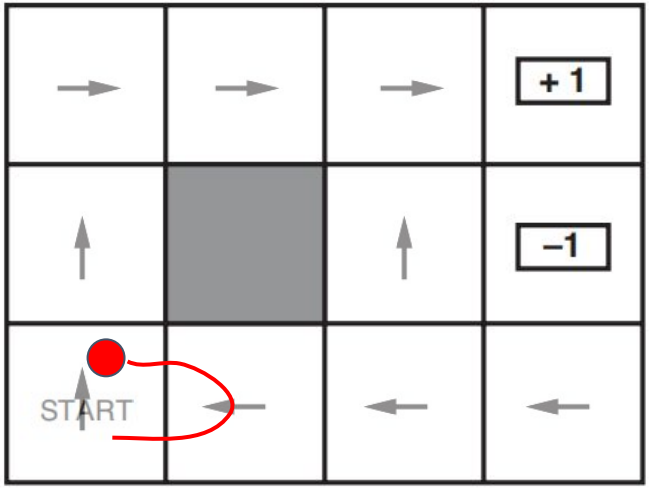
1. take action LEFT at (2,1), end up moving left.

2. (1,1) is not newly visit, do nothing

3. update utility at (2,1)

U(2,1) □ U(2,1) + α { R(2,1) + γ U(1,1) - U(2,1) }

U(1,1)	U(2,1)	U(1,2)
r + α*γ*r	r + α*γ*(r+α*γ*r)	0



Iteration 3

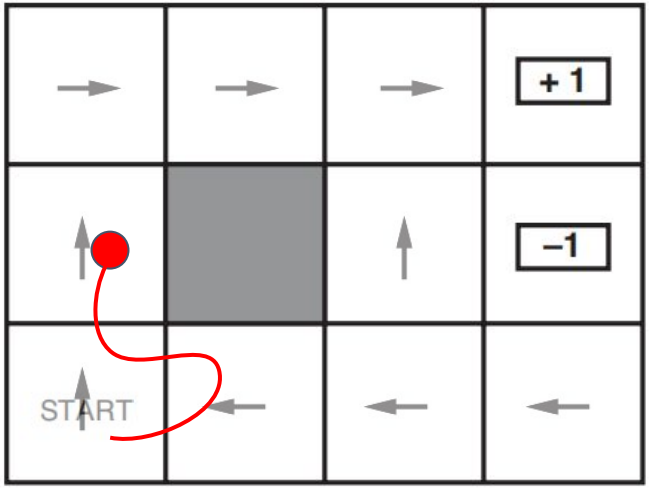
$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

1. take action UP at (1,1), end up moving up.

2. newly visit (1,2), set R(1,2)=r and U(1,2)=r

3. update utility at (1,1)

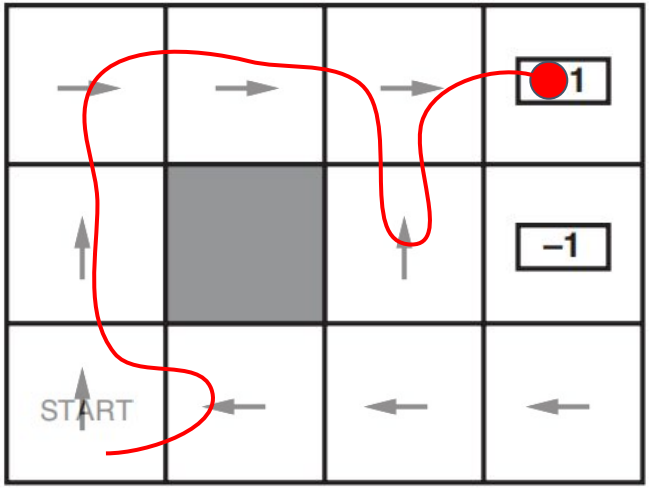
U(1,1) □ U(1,1) + α { R(1,1) + γ U(1,2) - U(1,1) }



U(1,1)	U(2,1)	U(1,2)
r + 2αγr - $\alpha^2\gamma r$	r + α*γ*(r+α*γ*r)	r

Explore until 1st trial terminates.

(1,1)	U(1,1) = r
(1,1) → (2,1)	U(1,1) □ U(1,1) + α {R(1,1)+γ*U(2,1)-U(1,1)}
(2,1) → (1,1)	U(2,1) □ U(2,1) + α {R(2,1)+γ*U(1,1)-U(2,1)}
(1,1) → (1,2)	U(1,1) □ U(1,1) + α {R(1,1)+γ*U(1,2)-U(1,1)}
(1,2) → (1,3)	U(1,2) □ U(1,2) + α {R(1,2)+γ*U(1,3)-U(1,2)}
(1,3) → (2,3)	U(1,3) □ U(1,3) + α {R(1,3)+γ*U(2,3)-U(1,3)}
……	……

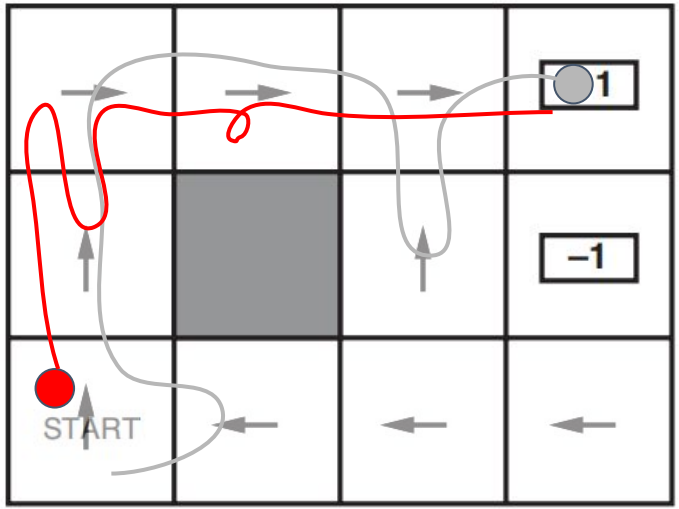


2nd Trial of Exploration

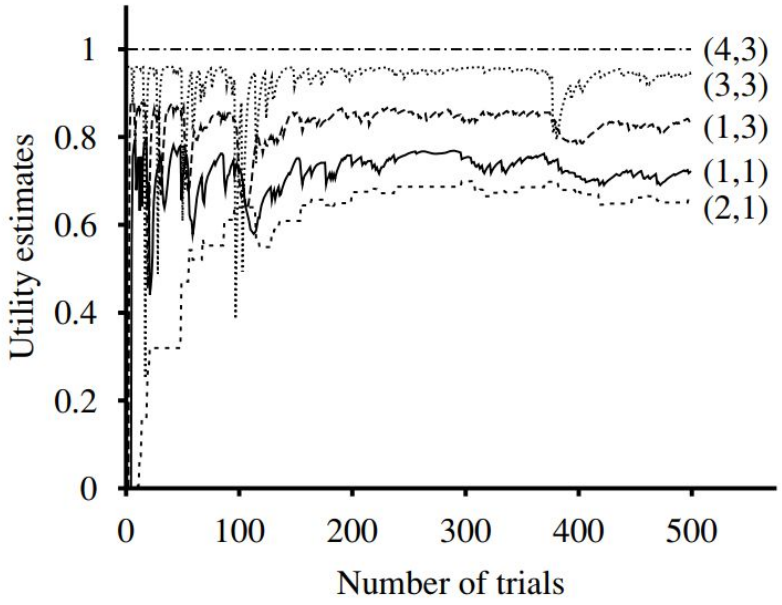
(1,1)	U(1,1) is result from 1st trial.
(1,1) → (1,2)	U(1,1) ← U(1,1) + α {R(1,1)+γ*U(1,2)-U(1,1)}
(1,2) → (1,3)	U(1,2) ← U(1,2) + α {R(1,2)+γ*U(1,3)-U(1,2)}
……	……

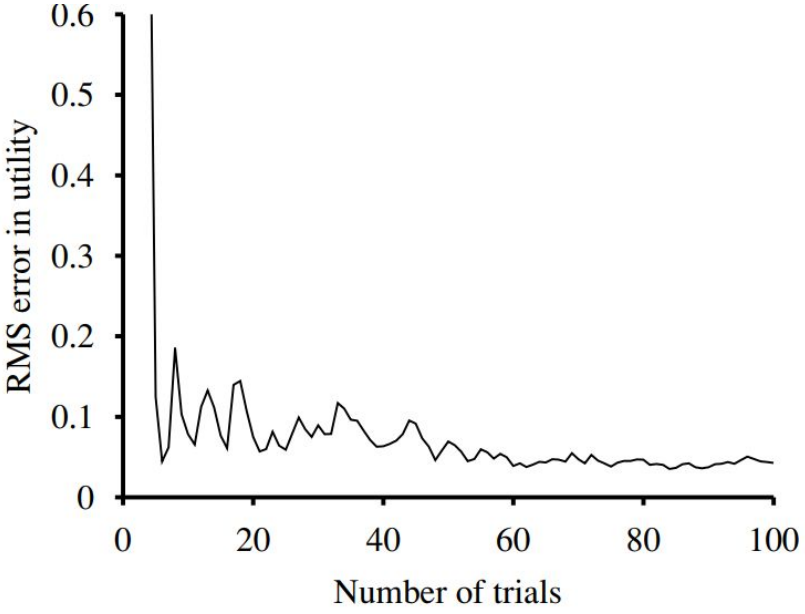
Average estimates of multiple trails to get the utility estimate for (1,1).

Then, apply this to every state.



Simulation Results





Active RL Solution: Q-Learning

It doesn’t have a given policy (as in passive RL), and needs to identify the optimal policy while exploring the world.

We may approach this problem by estimating the utility at each state based on different actions, and then pick the action that results in the maximum utility.

The utility at each state based on an action is measured by “Q” function, also known as the action-value (or action-utility) function.

The basic idea of Q-learning is to, with a randomly initialized policy, continuously update Q(s,a) right after visiting s and taking action a.

Action a is chosen based on the current policy; “a” with higher Q(s,a) will be chosen with higher probability but not probability 1 (the exploration-exploitation strategy).

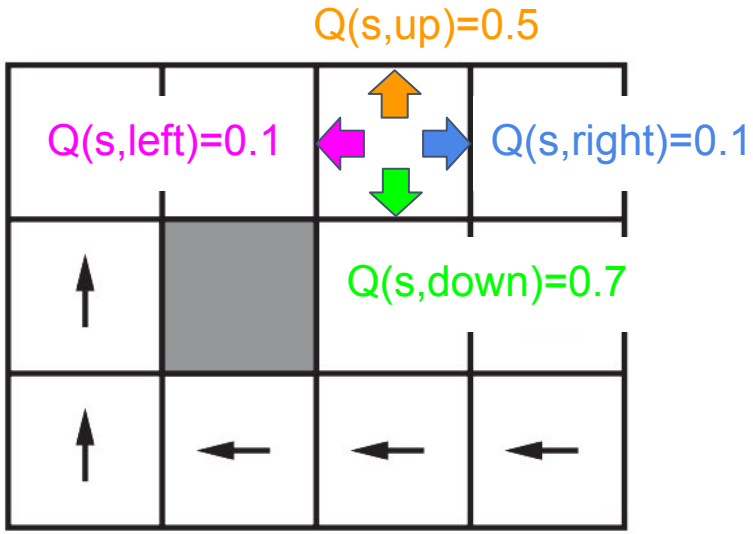
Action-Utility Function $Q_\pi(s,a)$

$Q_\pi(s,a)$ is the expected total reward for acting "a" at "s" (and follow π thereafter).

$$Q_\pi(s,a) = R(s) + \gamma \sum_{s'} p(s' \mid s,a) \cdot U_\pi(s')$$

reward at the current state

expected total reward from the next state



Example

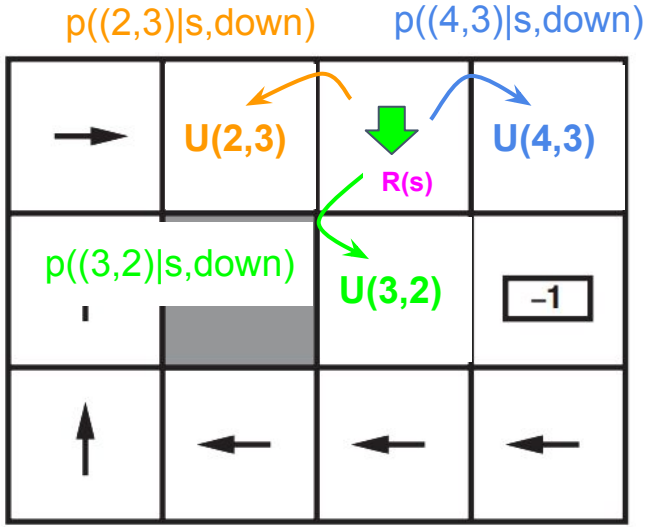
$Q_\pi(s,a)$ is the expected total reward for acting “a” at “s” (and follow π thereafter).

$$Q_\pi(s,a) = R(s) + \gamma \sum_{s'} p(s' \mid s,a) \cdot U_\pi(s')$$

$Q_\pi((3,3),\text{down}) = R(3,3) + \gamma * [\, p((2,3)|s,\text{down}) * U(2,3)$

$+ p((4,3)|s,\text{down}) * U(4,3)$

$+ p((2,3)|s,\text{down}) * U(2,3)]$



Estimate Q(s,a)

We can apply Temporal Difference Learning (TDL) to estimate Q(s,a).

All we need is to replace U(s) in TDL with Q(s,a).

Specifically, if the agent moves from s to s' after taking action a, update

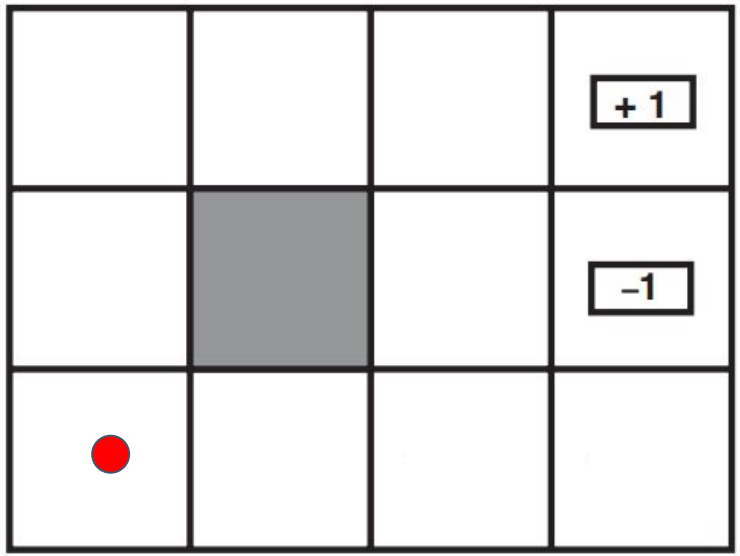
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Example

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Start at s = (1,1). Set R(1,1) = r.

Randomly initialize all Q's (or, just set to 0).



	UP	DOWN	LEFT	RIGHT
Q((1,1),a)	0	0	0	0
Q((2,1),a)	0	0	0	0

Iteration 1

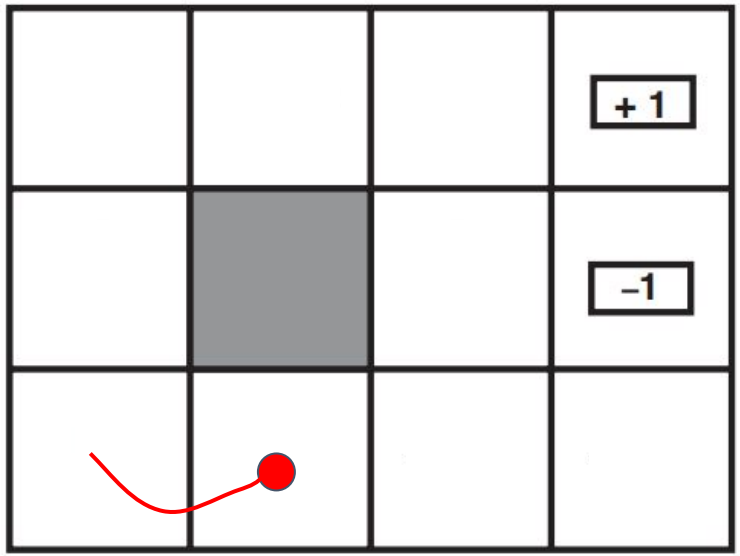
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Take UP at (1,1), but move to (2,1); set R(2,1)=r.

Update Q((1,1),up) = Q((1,1),up) + α*Δ

Δ = R(1,1) + γ*max$_{a'}$Q((2,1),a') - Q((1,1),up)

	UP	DOWN	LEFT	RIGHT
Q((1,1),a)	α*r	0	0	0
Q((2,1),a)	0	0	0	0



Iteration 2

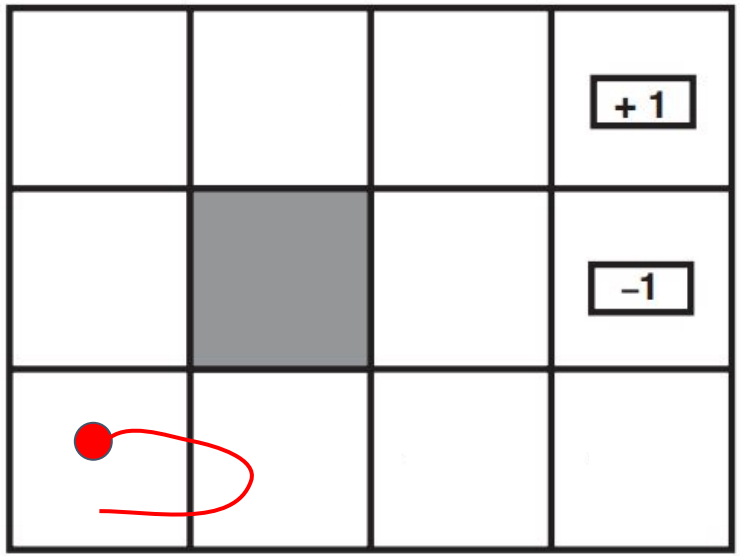
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Take LEFT at (2,1), move to (1,1).

Update Q((2,1),left) = Q((2,1),left) + α*Δ

Δ = R(2,1) + γ*max$_{a'}$Q((1,1),a') - Q((2,1),left)

	UP	DOWN	LEFT	RIGHT
Q((1,1),a)	α*r	0	0	0
Q((2,1),a)	0	0	α*(r+γ*α*r)	0



Iteration 3

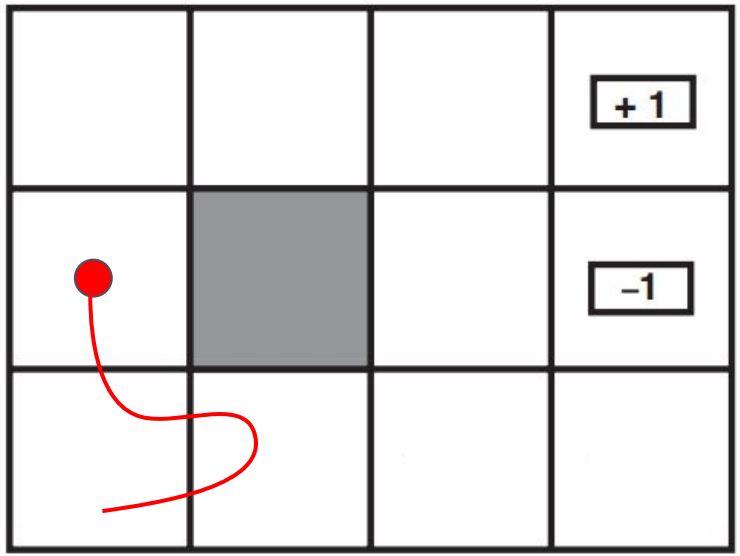
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Take RIGHT at (1,1), but move to (1,2). Set R(1,2)=r.

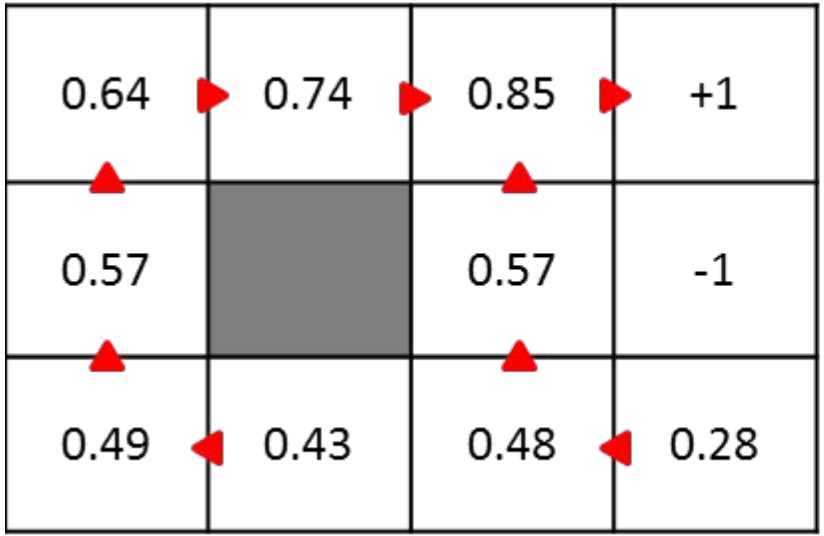
Update Q((1,1),right) = Q((1,1),right) + α*Δ

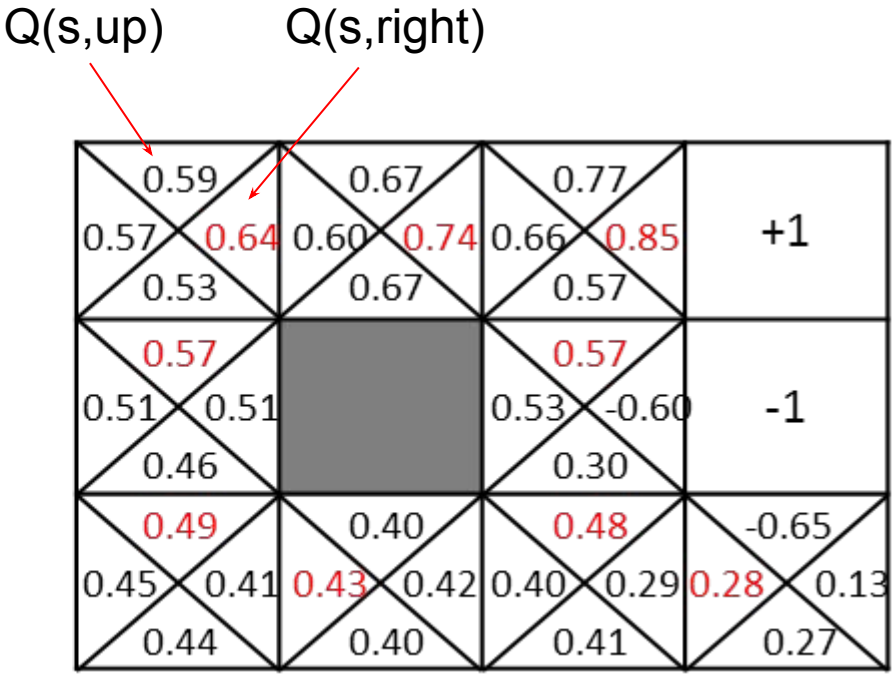
Δ = R(1,1) + γ*max$_{a'}$Q((1,2),a') - Q((1,1),right)

	UP	DOWN	LEFT	RIGHT
Q((1,1),a)	α*r	0	0	α*r
Q((2,1),a)	0	0	α*(r+γ*α*r)	0



Example Results





Q-Learning Algorithm

1. Put agent at s = s_0 and randomly initialize all Q's. Set learning rate α.

2. Agent chooses an action *a* at state s, and moves to s'.

3. If s' is newly visited, set R(s') = r and U(s') = r.

4. Update Q(s,a) by

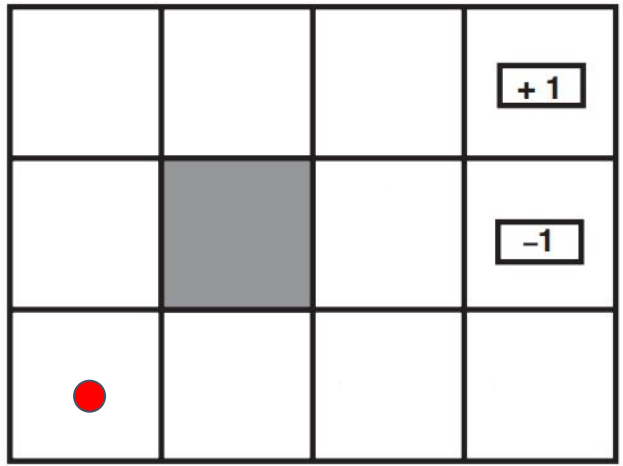
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

5. Update optimal policy π(s) = $\max_{a'} Q(s,a')$ and s = s'

6. Repeat 2-5 until convergence.

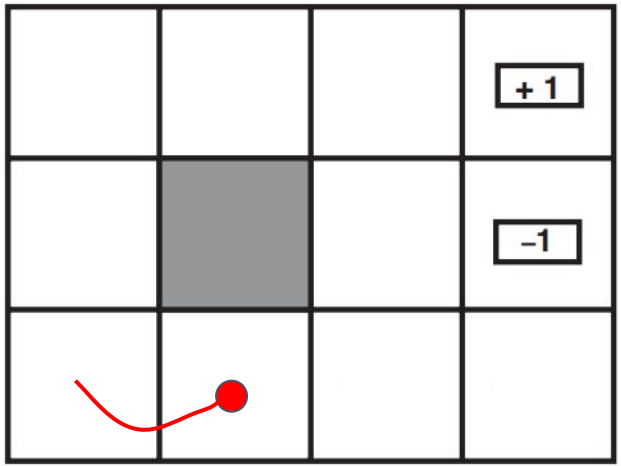
Example

	UP	DOWN	LEFT	RIGHT	π(s)
Q((1,1),a)	0	0	0	0	/
Q((2,1),a)	0	0	0	0	/



Iteration 1

	UP	DOWN	LEFT	RIGHT	π(s)
Q((1,1),a)	0	0	0	0.6	right
Q((2,1),a)	0	0	0	0	/



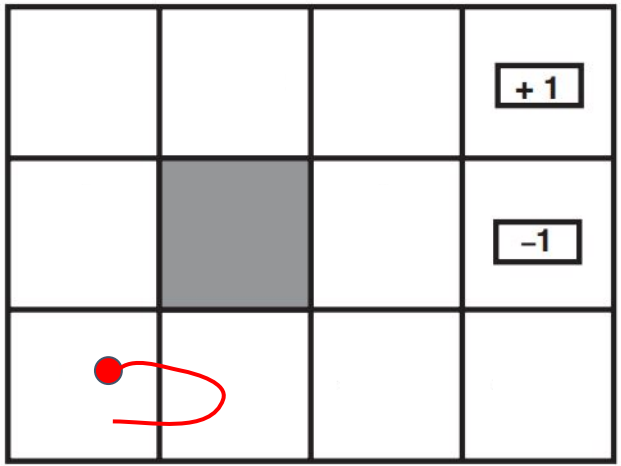
Randomly choose RIGHT, move RIGHT.

Update Q((1,1), right).

Update π((1,1)).

Iteration 2

	UP	DOWN	LEFT	RIGHT	π(s)
Q((1,1),a)	0	0	0	0.6	right
Q((2,1),a)	0.4	0	0	0	up



Randomly choose UP, end up moving LEFT.

Update Q((2,1),up).

Update π((2,1)).

Iteration 3

	UP	DOWN	LEFT	RIGHT	π(s)
Q((1,1),a)	0.7	0	0	0.6	up
Q((2,1),a)	0.4	0	0	0	up

With 1-p_{π}, randomly choose UP, end up moving UP.

Update Q((1,1),up).

Update π((1,1)).