

A hand holding a bone over a dog. The background is a gradient from dark grey to white. A hand from the top left holds a yellow bone. A white dog with brown patches and a blue collar is lying down on the right side, looking up at the bone.

# CS 5/7320

## Artificial Intelligence

### Reinforcement Learning

#### AIMA Chapter 17+22

Slides by Michael Hahsler  
with figures from the AIMA textbook.



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

# Remember Chapter 16:

## Making Simple Decisions

For a decision that we make frequently and making it once does not affect the future decisions (**episodic environment**), we can use the **Principle of Maximum Expected Utility (MEU)**.

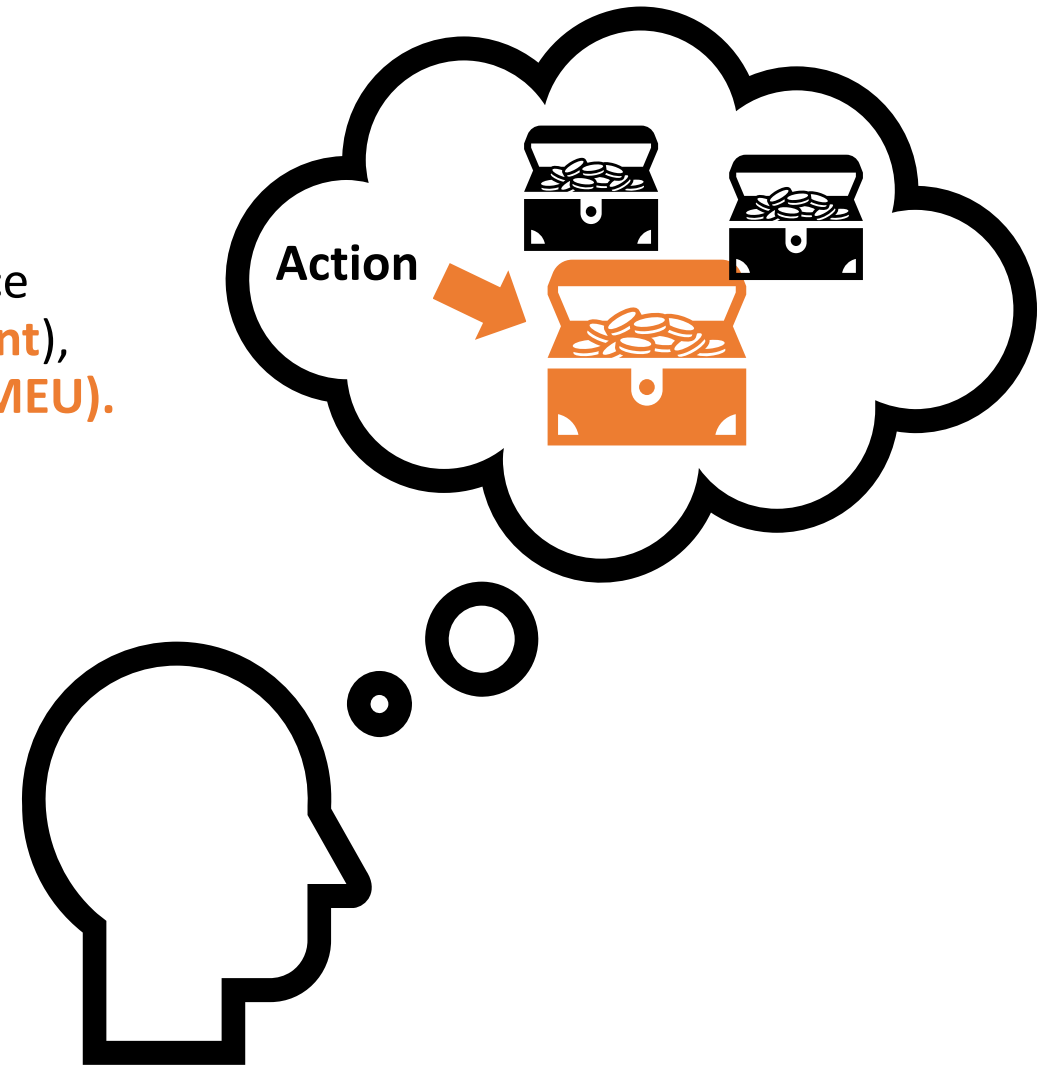
Given the expected utility of an action

$$EU(a) = \sum_{s'} P(Result(a) = s')U(s')$$

choose action that maximizes the expected utility:

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

Now we will talk about decision making in **sequential environments**.



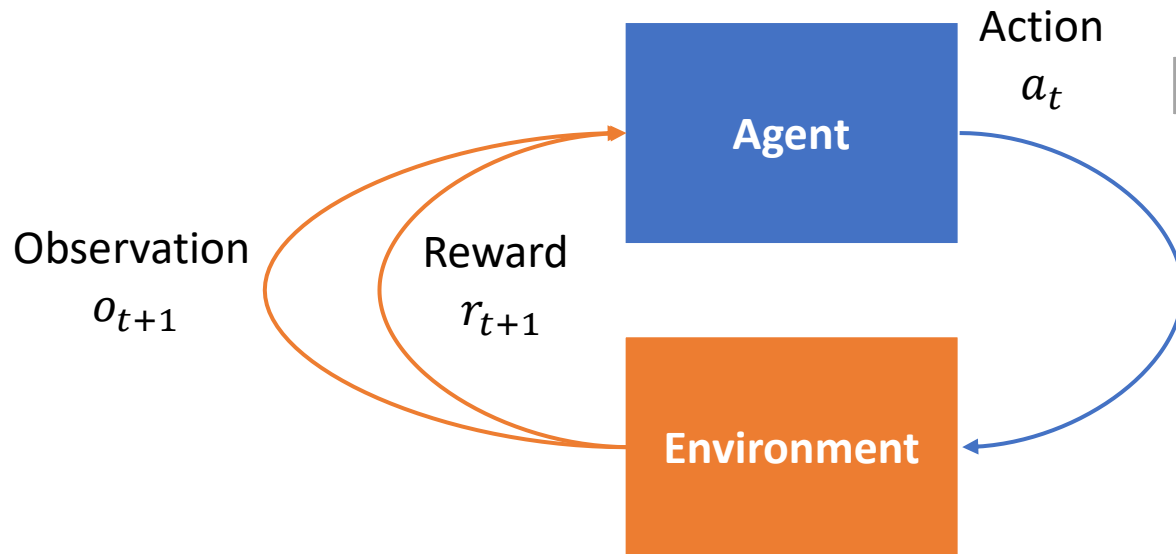


# Sequential Decision Problems

AIMA Chapter 17: Making Complex Decisions

# Sequential Decision Problems

- **Utility-based agent:** The agent's utility depends on a sequence of decisions that depend on each other.
- Sequential decision problems incorporate utilities, uncertainty, and sensing.



Sequence:  $(o_0, r_0), a_0, (o_1, r_1), a_1, (o_2, r_2), a_2, \dots$

**Goal:** Observations and rewards depend on the state of the system and the agent wants to maximize the expected discounted reward:

$$U = E \left[ \sum_{t=0}^T \gamma^t r_t \right]$$

$\gamma$  ... discounting factor

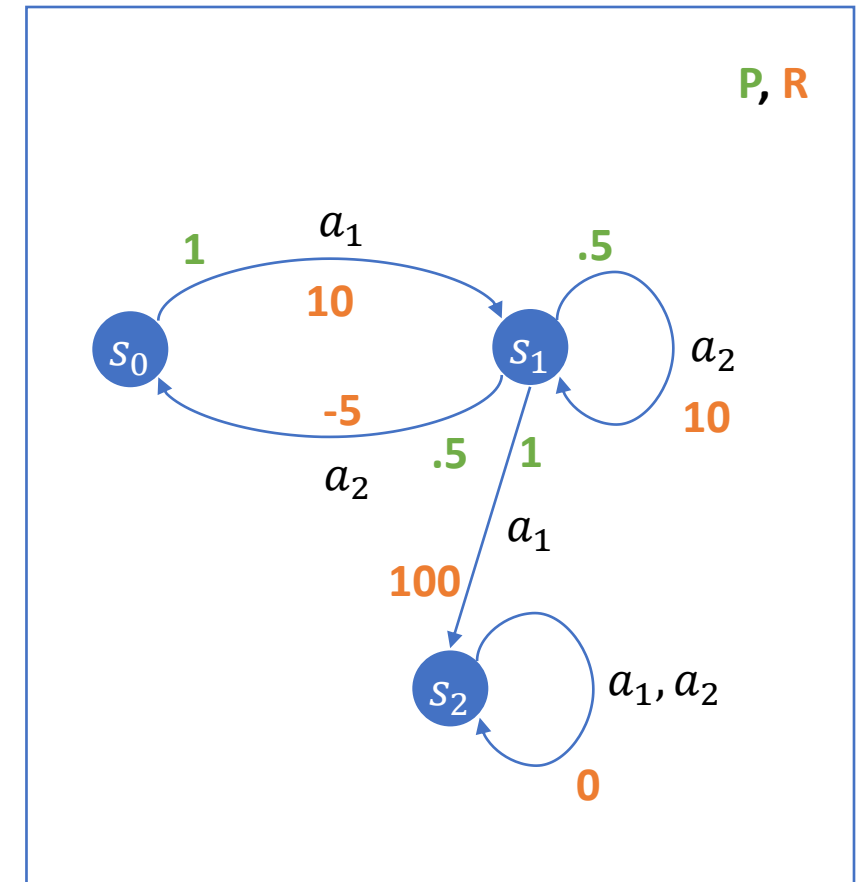
$T$  ... time horizon may be infinity



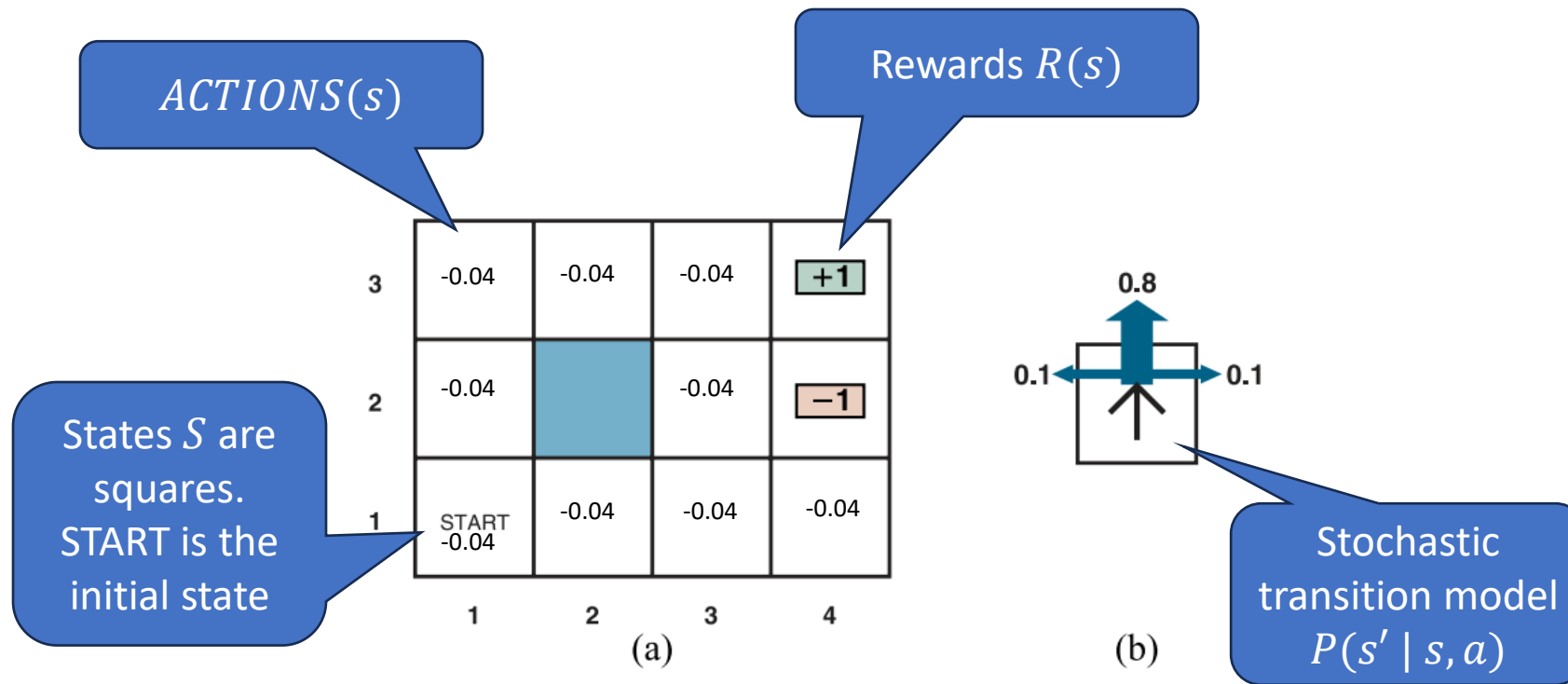
# Definition: Markov Decision Process (MDP)

- MDPs are sequential decision problems with
  - a fully observable ( $o_t = s_t$ ), stochastic environment;
  - a Markovian transition model: future states do not depend on past states given the current state;
  - additive rewards.
- MDPs are discrete-time stochastic control processes defined by:
  - a finite set of **states**  $S = \{s_0, s_1, s_2, \dots\}$  (initial state  $s_0$ )
  - a set of available **actions**  $ACTIONS(s)$  in each state  $s$
  - a **transition model**  $P(s' | s, a)$  where  $a \in ACTIONS(s)$
  - a **reward function**  $R(s)$  where the reward depends on the current state (often  $R(s, a, s')$  is used)
- Time horizon
  - **Infinite horizon**: non-episodic (continuous) tasks with no terminal state.
  - **Finite horizon**: episodic tasks. Episode ends after a number of periods or when a terminal state is reached. Episodes contain a sequence of several actions that affect each other.

This is different from the previous definition of an **episodic** environment!



# Example: 4x3 Grid World



## Goal

For each square: determine what direction should we try to go to maximize the total utility?

This is called a **policy** written as the function

$$\pi: S \rightarrow ACTIONS(S)$$

**Figure 17.1** (a) A simple, stochastic  $4 \times 3$  environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the “intended” outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and -1, respectively, and all other transitions have a reward of -0.04.

# Optimal Policy

- A policy  $\pi = \{\pi(s_0), \pi(s_1), \dots\}$  defines for each state which action to take.
- The expected utility of being in state  $s$  under policy  $\pi$  can be calculated as the sum:

$$U^\pi(s) = E_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s \right]$$

$\gamma$  is a discounting factor to give more weight to immediate rewards.

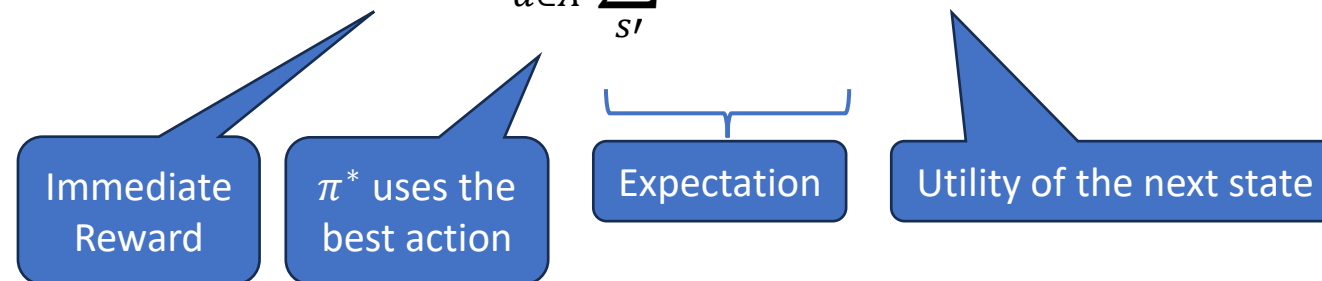
$E_\pi$  is the expectation over sequences that can be created by following  $\pi$ .

- The goal of solving an MDP is to find an optimal policy  $\pi$  that maximizes the expected future utility for each state

$$\pi^*(s) = \operatorname{argmax}_{\pi} U^\pi(s) \quad \text{for all } s \in S$$

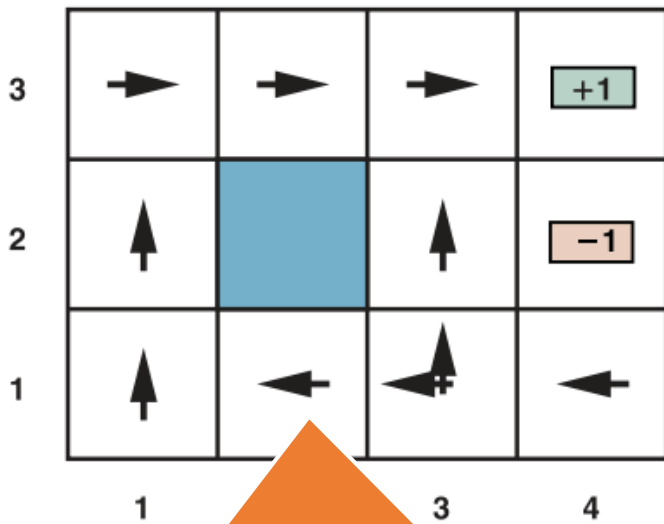
- The recursive **Bellman equation** holds for the optimal value function  $U$  (“Bellman optimality condition”):

$$U^{\pi^*}(s) = R(s) + \gamma \max_{a \in A} \sum_{s'} P(s'|s, a) U^{\pi^*}(s')$$



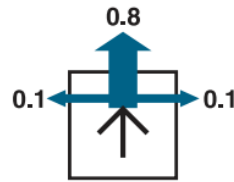
# Solution: 4x3 Grid World

Optimal action in each state  
(policy  $\pi^*$ )

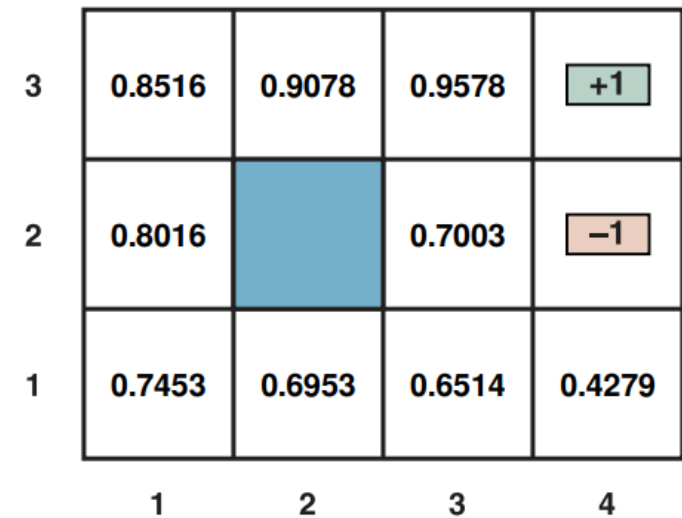


Why is it optimal to walk away from the +1 square?

Always pick the action with the highest expected utility.



Value of being in a state  $U(s)$   
(given that we will follow  $\pi^*$ )



$\gamma = 1$

How to we find the optimal value function/optimal policy?

Policy Iteration

Value Iteration



# Q-Function

- $Q(s, a)$  is called the state-action value function. It gives the expected utility of action  $a$  in state  $s$ .

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) [U(s')]$$



Immediate  
Reward

The diagram shows the equation  $Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) [U(s')]$  with two blue callout boxes. The first box, labeled 'Immediate Reward', points to the  $R(s)$  term. The second box, labeled 'Expected utility of the next state', points to the  $U(s')$  term inside the summation.

Expected utility of the  
next state

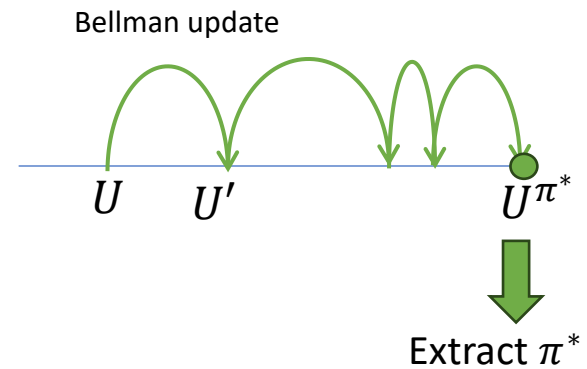
- Relationship with the state value function:  $U(s) = \max_a Q(s, a)$
- The Q-function is often used for convenience in algorithms.

# Value Iteration: Estimate the Optimal Value Function $U^{\pi^*}$

**Algorithm:** Start with a  $U$  vector of 0 for all states and then update (Bellman update) the vector iteratively until it converges to the unique optimal solution  $U^{\pi^*}$ .

```
function VALUE-ITERATION( $mdp, \epsilon$ ) returns a utility function
  inputs:  $mdp$ , an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ ,
           rewards  $R(s, a, s')$ , discount  $\gamma$ 
            $\epsilon$ , the maximum error allowed in the utility of any state
  local variables:  $U, U'$ , vectors of utilities for states in  $S$ , initially zero
                      $\delta$ , the maximum relative change in the utility of any state

  repeat
     $U \leftarrow U'; \delta \leftarrow 0$ 
    for each state  $s$  in  $S$  do
       $U'[s] \leftarrow \max_{a \in A(s)} \text{Q-VALUE}(mdp, s, a, U)$ 
      if  $|U'[s] - U[s]| > \delta$  then  $\delta \leftarrow |U'[s] - U[s]|$ 
  until  $\delta \leq \epsilon(1 - \gamma)/\gamma$ 
  return  $U$ 
```



Update with the value of the best action in state  $s$ .

Uses a proxy for policy loss  $\|U^{\pi} - U\|_{\infty}$  as the stopping criterion

$U$  converges to  $U^{\pi^*}$  and we can extract  $\pi^*$

# Policy Iteration: Learn the Optimal Policy $\pi^*$

Policy iteration tries to directly find the optimal policy by iterating policy evaluation and improvement.

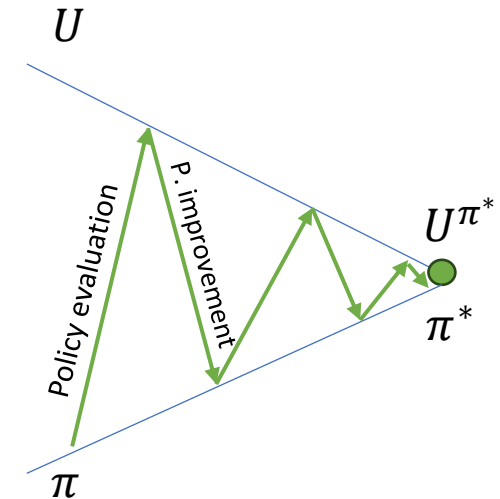
```
function POLICY-ITERATION(mdp) returns a policy
inputs: mdp, an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ 
local variables:  $U$ , a vector of utilities for states in  $S$ , initially zero
                   $\pi$ , a policy vector indexed by state, initially random

repeat
   $U \leftarrow \text{POLICY-EVALUATION}(\pi, U, \text{mdp})$ 
  unchanged?  $\leftarrow$  true
  for each state  $s$  in  $S$  do
     $a^* \leftarrow \operatorname{argmax}_{a \in A(s)} \text{Q-VALUE}(\text{mdp}, s, a, U)$ 
    if  $\text{Q-VALUE}(\text{mdp}, s, a^*, U) > \text{Q-VALUE}(\text{mdp}, s, \pi[s], U)$  then
       $\pi[s] \leftarrow a^*$ ; unchanged?  $\leftarrow$  false
  until unchanged?
return  $\pi$ 
```

Calculate  $U$  given current policy  
(either solve an LP or value  
iteration with fixed policy)

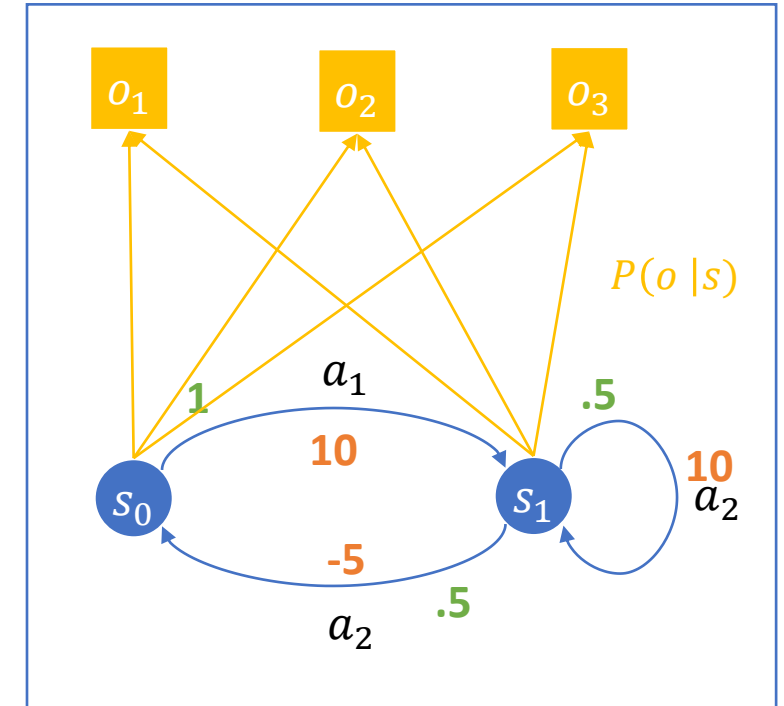
Greedy policy  
Improvement

$\pi$  converges to  $\pi^*$   
(and  $U$  converges to  $U^{\pi^*}$ )



# Partially Observable Markov Decision Model (POMDP)

- If the environment is **partially observable**, then  $o_t \neq s_t$  and the model is expanded by
  - a **sensor model**  $P(o | s)$  for receiving observation  $o$  given being in state  $s$ .
- This makes things a lot more complicated, and we have to work with **belief states**. A belief state is a distribution over states.  
Example: For a problem with three states, the belief state  $b = (.2, .8, 0)$  means the agent beliefs that it is with 20% in state 1 and 80% in state 2 but not in state 3.
- An MDP that uses belief states instead of system states is called a **belief MDP**.  
Issue: the probabilities in belief states are continuous, and the number of different belief states is infinite.
- The solution of a POMDP is a policy with the optimal actions for sets of belief states (i.e., ranges of belief).
- For all but tiny problems, POMDPs can only be solved **approximately** (e.g., by grid-based methods).





# Reinforcement Learning

AIMA Chapter 22

# Reinforcement Learning (RL)

- RL assumes that the problem can be modeled by an **MDP**.
- What if we do not know the exact transition model  $P(s' \mid s, a)$ ?

Now we cannot solve the MDP (estimate the state utility function/policy) because we cannot predict what the future states after an action will be!

- The agent needs to explore (try actions) and **use the reward signal to update its estimate of the utility of states and actions**. This is a learning process where the reward provides positive reinforcement.



# Q-Learning

- Q-Learning learns the state-action value function from interactions with the environment (percepts).
- This agent function learns a **Q-table** for the state-action function  $Q$ .

Q-Table

$s$	$a$	$Q(s, a)$

**function** Q-LEARNING-AGENT(*percept*) **returns** an action

**inputs:** *percept*, a percept indicating the current state  $s'$  and reward signal  $r$

**persistent:**  $Q$ , a table of action values indexed by state and action, initially zero

$N_{sa}$ , a table of frequencies for state–action pairs, initially zero

$s, a$ , the previous state and action, initially null

New episode  
has no  $s$ .

**if**  $s$  is not null **then**

increment  $N_{sa}[s, a]$

$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

$s, a \leftarrow s', \operatorname{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$

**return**  $a$

Learning rate

Make  $Q[s, a]$  a little more similar to the received reward + the best Q-value of the successor state.

$f$  is the exploration function and decides on the next action. As  $N$  increases it can exploit good actions more.

# Value Function Approximation

- $U(Q)$  needs to store and estimate one entry for each state (state/action combination).
- Issues and solutions
  - Too many entries to store → lossy compression
  - Many combinations are rarely seen → generalize to unseen entries
- **Idea:** Estimate the state value by learning an approximation function  $\hat{U}(s) = g_{\theta}(s)$  based on features of  $s$ .
- **Example:** 4x3 Grid World with a linear combination of state features  $(x, y)$  and learn  $\theta$  from observed data.

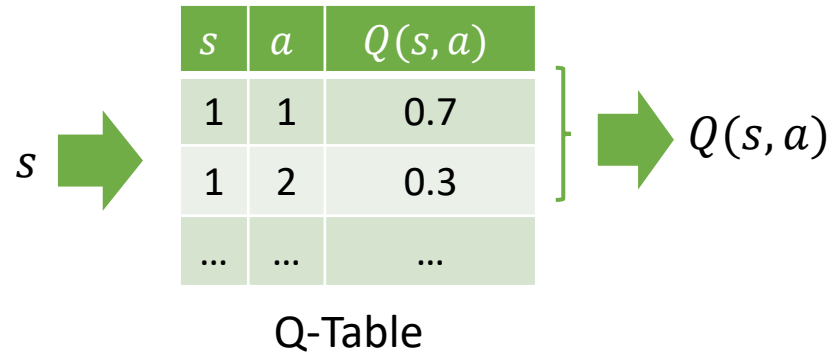
3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	-1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

Learn  $\theta$  from observed interactions with the environment to approximate  $U(s)$

$$\hat{U}_{\theta}(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

$\theta$  can be updated iteratively after each new observed utility using gradient descent.

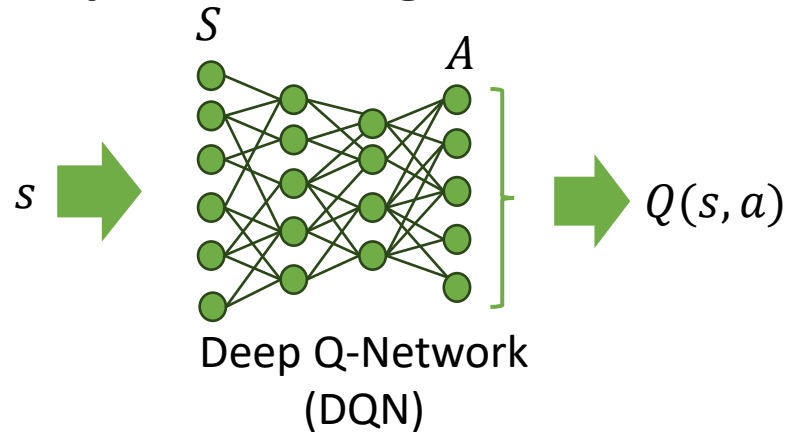
## Traditional Q-Learning



**function** Q-LEARNING-AGENT(*percept*) **returns** an action  
**inputs:** *percept*, a percept indicating the current state  $s'$  and reward signal  $r$   
**persistent:**  $Q$ , a table of action values indexed by state and action, initially zero  
 $N_{sa}$ , a table of frequencies for state–action pairs, initially zero  
 $s, a$ , the previous state and action, initially null

**if**  $s$  is not null **then**  
 increment  $N_{sa}[s, a]$   
 $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(\underbrace{r + \gamma \max_{a'} Q[s', a']}_{\text{target}} - \underbrace{Q[s, a]}_{\text{prediction}})$   
 $s, a \leftarrow s', \operatorname{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$   
**return**  $a$

## Deep Q-Learning



**Target networks:** It turns out that the Q-Network is unstable if the same network is used to estimate  $Q(s, a)$  and also  $Q(s', a')$ . Deep Q-Learning uses a second target network for  $Q(s', a')$  that is updated with the prediction network every  $C$  steps.

**Experience replay:** To reduce instability more, generate actions using the current network and store the experience  $\langle s, a, r, s' \rangle$  in a table. Update the model parameters by sampling from the table.

**Loss function:** squared difference between prediction and target.



# Summary

- Agents can learn the value of being in a state from **reward signals**.
- Rewards can be delayed (e.g., at the end of a game).
- Not being able to fully **observe the state** makes the problem more difficult (POMDP).
- **Unknown transition models** lead to the need of exploration by trying actions (model free methods like Q-Learning).
- All these problems are computationally very expensive and often can only be solved by **approximation**. State of the art is to use deep artificial neural networks for function approximation.