
Reinforcement Learning (I)

Markov Decision Process

Outline

- **Introduction**
- **Basic concepts**
- **Markov decision processes (MDPs)**
 - **Markov chains (basic concepts)**
 - **Markov decision processes (MDPs)**
 - **Planning in MDPs**
 - **Extensions to MDPs**

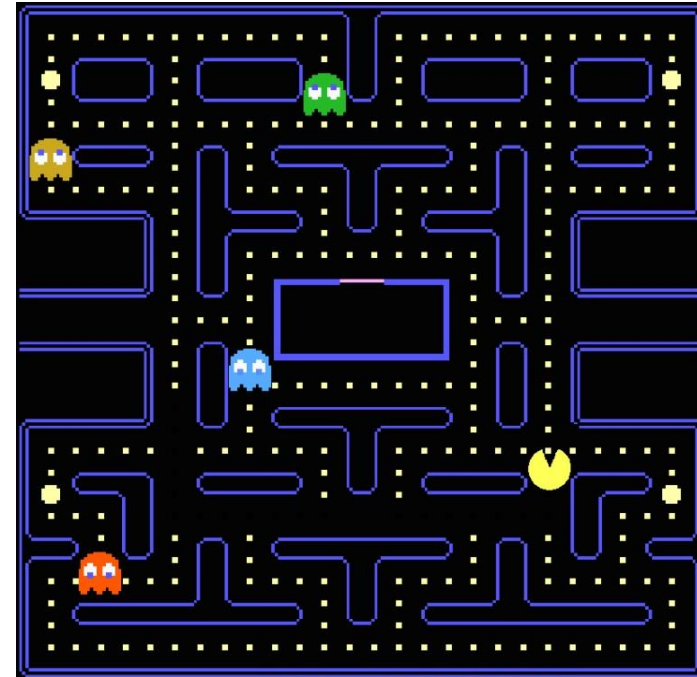
Introduction

How do you learn to play a game?

- One possible approach: Play it!



- Good outcome
 - did something right
 - do more

- Bad outcome
 - did something wrong
 - do less



Pacman developed by Toru Iwatani and released by Namco in 1980. (Philipp Rohlfshagen, Jialin Liu, Diego Perez-Liebana, and Simon M Lucas. Pac-man conquers academia : Two decades of research using a classic arcade game. IEEE Transactions on Games, 10(3) :233–256, 2018.)

What Did We Actually Do?

- **Play it!**  Interact with environment
 - **Good outcome**
 - did something right
 - do more
 - **Bad outcome**
 - did something wrong
 - do less
- 
- Learn from reward signals

Can we learn from this process?

Human Learning -> Computational Learning

- Human learn from **interaction** with the environment.
- Can we formulate some computational approach to learning from interactions?
=> Reinforcement Learning (RL).
- A RL **agent** learns how to map situations (scenarios/states) to actions aiming at maximizing a numerical reward.

Some Related Historical Notes

1. **Animal learning.**
 2. **Optimal control** and its solutions using
 - value functions;
 - and **dynamic programming**.
- **Reinforcement Learning** (late 1980s).

Optimal Control: Incomplete History

- “The problem of designing a controller to minimize a measure of a dynamical system’s behaviour over time.”
- **Bellman Equation**: uses the concepts of a dynamical system’s state and of a value function, or “optimal return function,” to define a functional equation (Hamilton and Jacobi -> Richard Bellman, mid-1950s).
- **Dynamic Programming**: the class of methods for solving optimal control problems by solving the Bellman Equation [Bellman, 1957].
- **Markov Decision Processes (MDPs)**: the discrete stochastic version of the optimal control problem introduced by Bellman.
- Ronald Howard (1960) devised the **policy iteration method** for MDPs.

Basic Concepts

Terminology

- **Environment:** We take optimal actions (hopefully) according to our **observations of the environment** and **predicted/estimated rewards** obtained by applying different sequences of legal actions.
- The **environment state** contains the information used to determine what happens next given an action/a sequence of actions.
- The **agent state** is the agent's representation of the observation or the environment, and contains the information used to determine what to act next.
- **Available action(s):** What an agent can do given a state -> State-dependent!
- **Goal:** One or more desirable states, such as “win in the game”.
- **Performance measure:** How to determine the solution quality. It usually includes a goal test, e.g., if we are in a goal state.

Environment State vs. Agent State

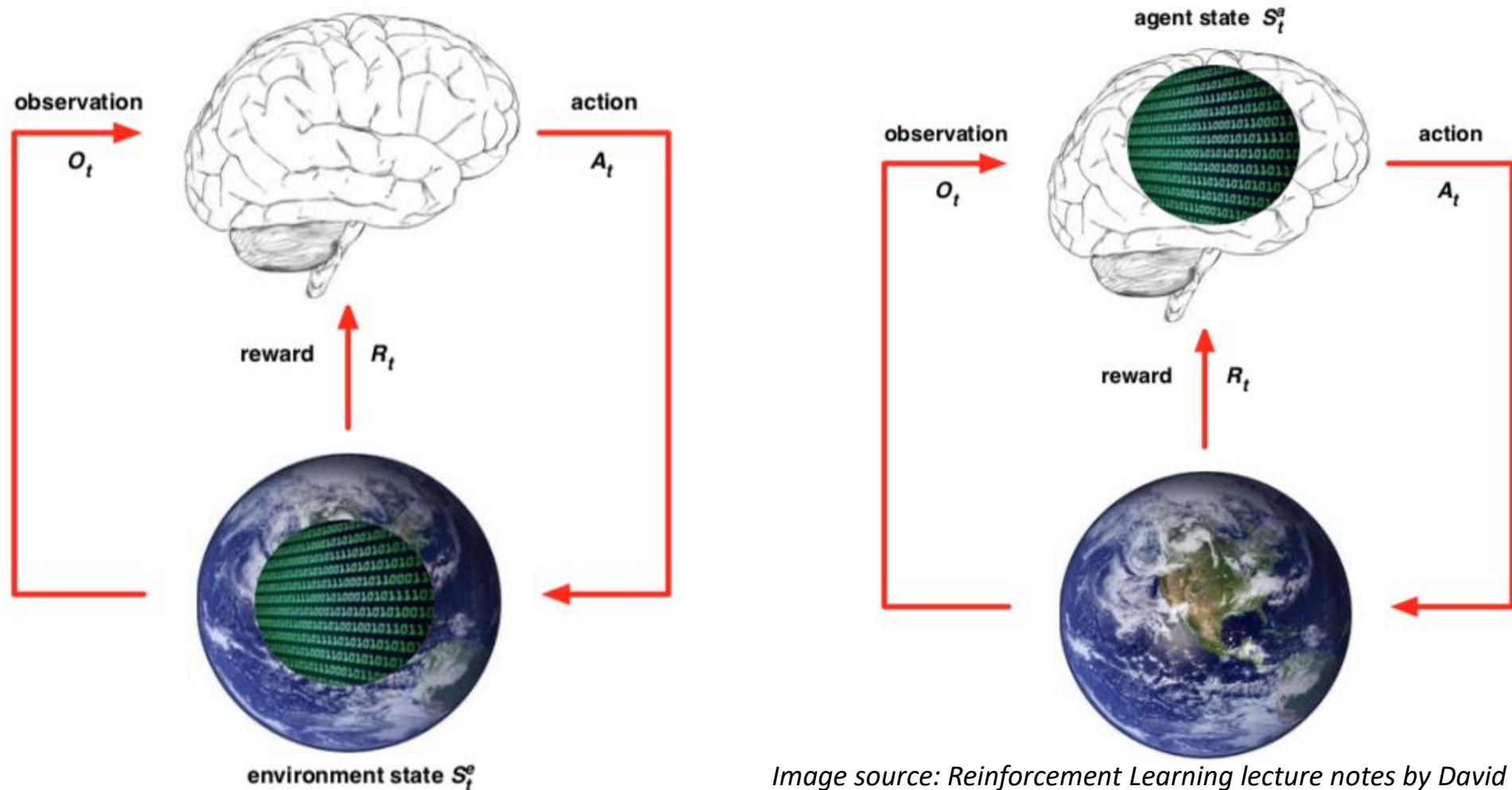


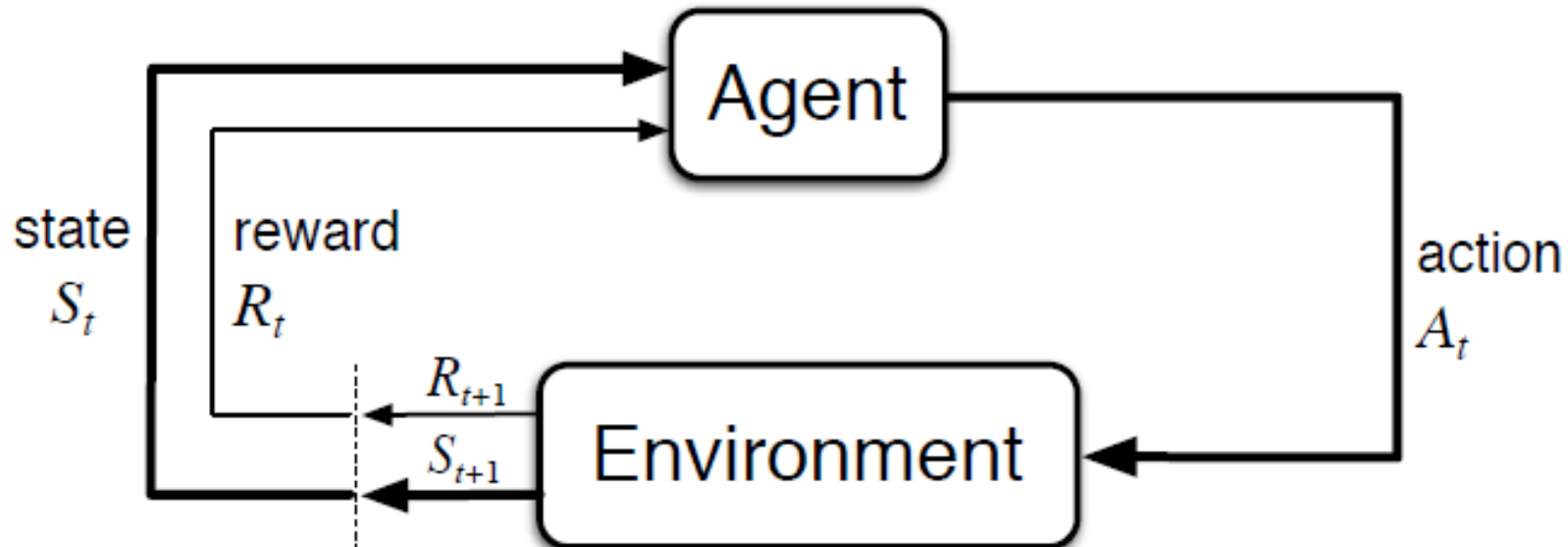
Image source: Reinforcement Learning lecture notes by David Silver.

State and Observation

- **The observation does not necessarily represent the full environment.**
- **Partial observability.**

Agent-Environment Interaction

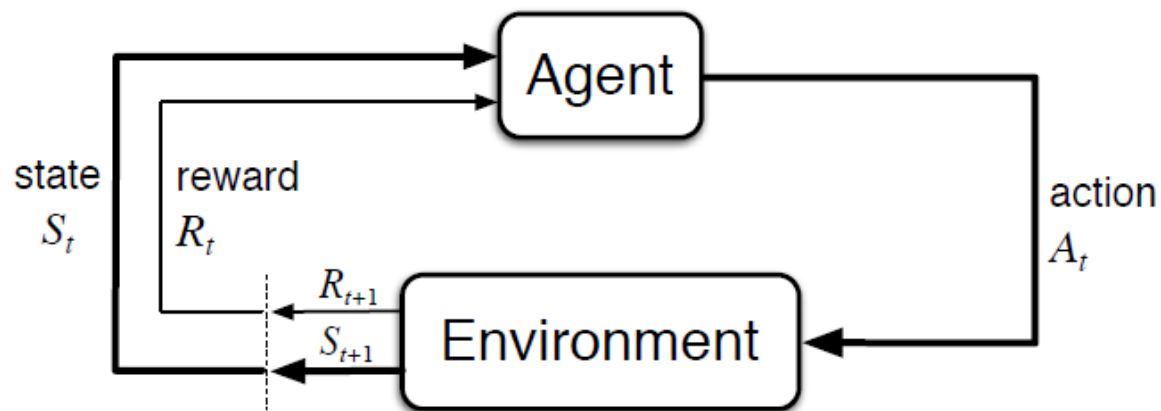
- An agent receives current state S_t and reward R_t from the environment, where t denotes the time step/tick.



Agent-Environment Interaction

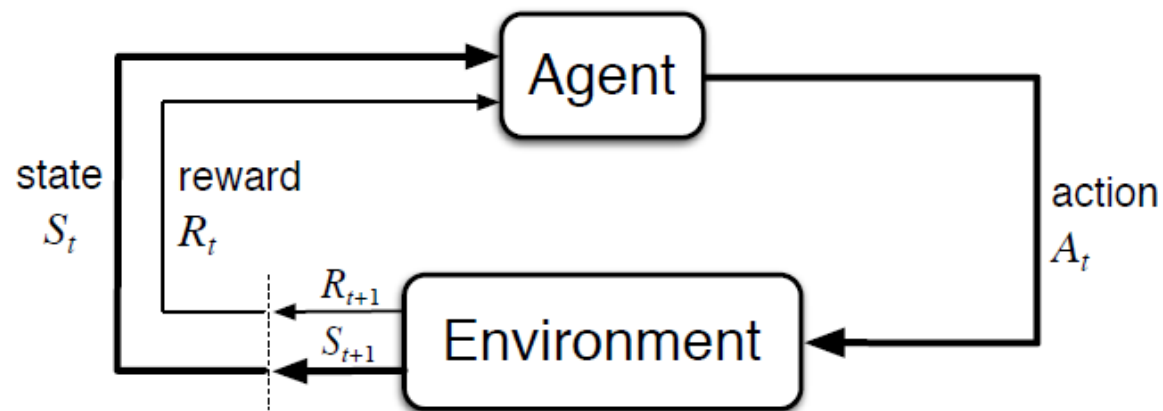
- Receive current state S_t and reward R_t from the environment.
- **Then take action A_t .**
- **Environment state and reward change to S_{t+1} and R_{t+1} (“transition”).**

$$\forall t, S_{t+1} = \mathcal{T}(S_t, A_t)$$



Reward Signals

- Reward R_t : a real number.
- Higher rewards \rightarrow better outcomes.
 - $R_t > 0$ does not necessarily indicates a good outcome .
 - Example: if all other rewards are +5 then +1 is not good.



Reward Hypothesis

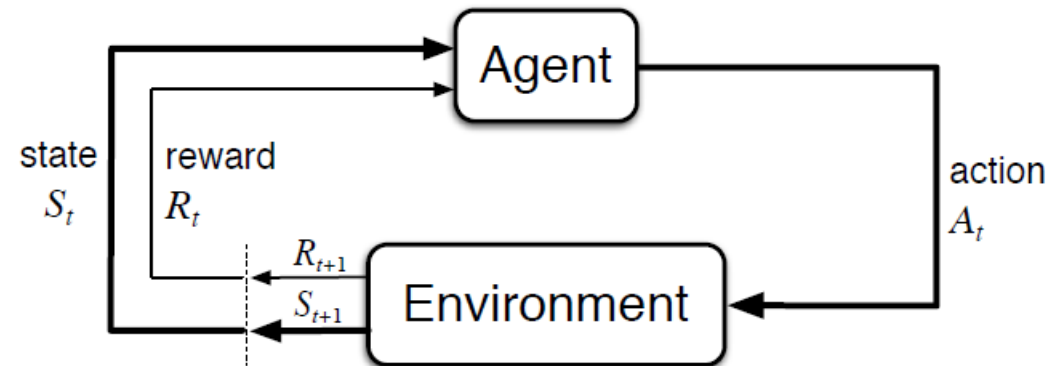
“That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).”

Objectives

- **Objective of interaction:** take actions that maximize the total reward (or “cumulative reward”).

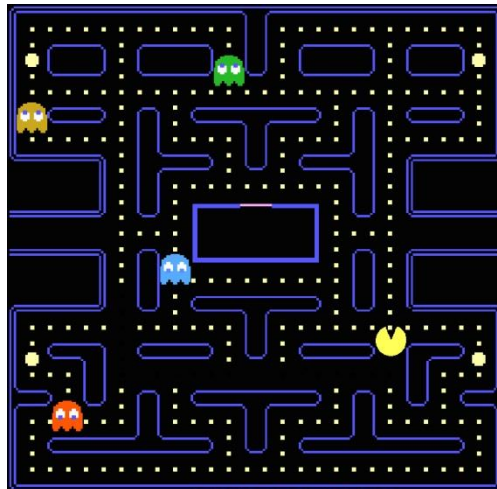
$$CR_T = \sum_{t=1}^T R_t$$

- **Objective of learning:** find out such actions from the information collected during interaction.



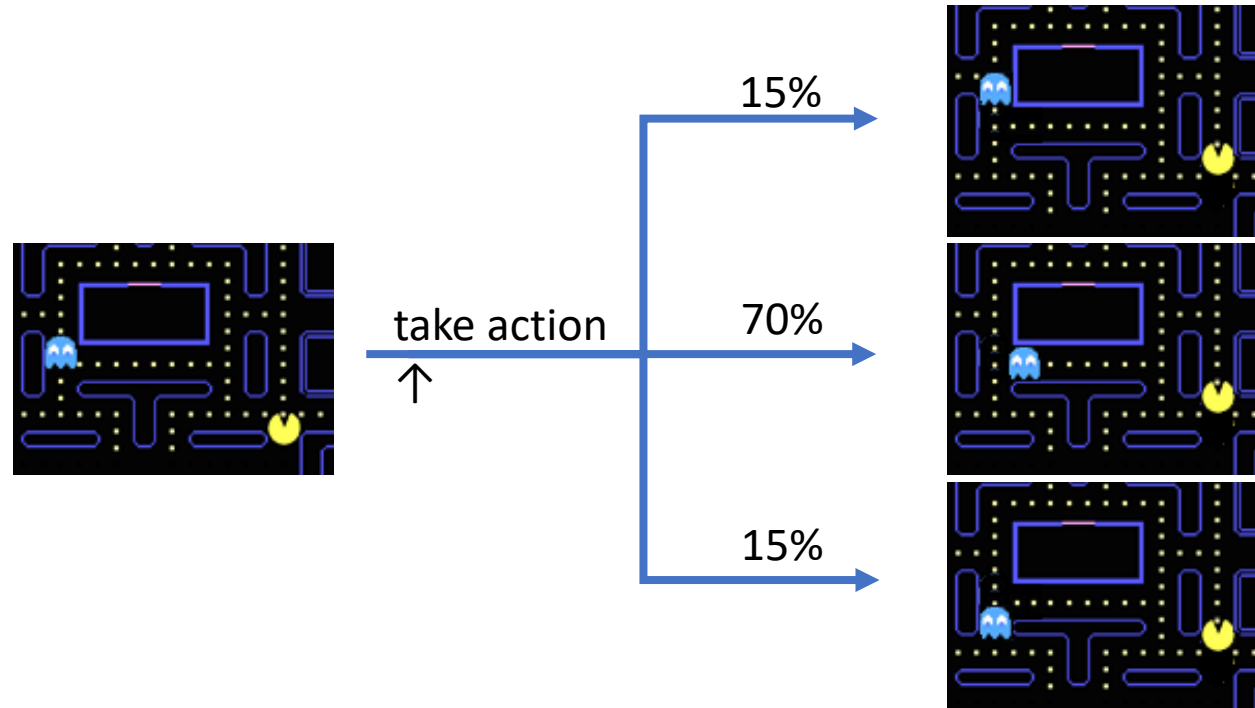
Example: Pacman

- States in *Pacman* = *<position of player, positions of ghosts, positions of items, powerup duration, ...>*
 - Should contain all information necessary for making decisions.



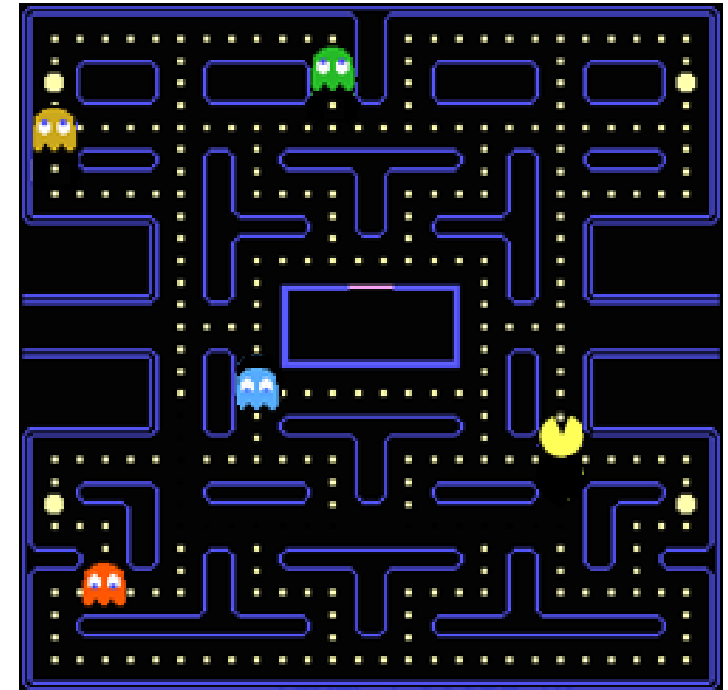
Example: Pacman

- Possible actions = $\{\uparrow, \downarrow, \leftarrow, \rightarrow, \text{stay still}\}$.
- State transition may involve randomness.



Example: Pacman

- Rewards in *Pacman*:
 - Eat a bean $\rightarrow +10$
 - Caught by ghosts $\rightarrow -100000$
 - Eat ghost whilst powerup $\rightarrow +200$
 - Otherwise $\rightarrow +0$
- Maximise cumulative reward
 \rightarrow eat more & avoid death.



Applications of RL

- **Game playing (Go, Atari 2600, StarCraft, LoL, Dota, Dota2, ...)**
- **Robotics**
- **Automated driving**
- **Power system control**
- **Automated stock trading**
- **.....**

Markov Decision Processes (MDPs)

- ❖ Markov chains (basic concepts)
- ❖ Markov decision processes (MDPs)
- ❖ Planning in MDPs
- ❖ Extensions to MDPs

Formulating RL problems

- 1. How to formulate transitions ($S_t \rightarrow S_{t+1}$) ?**
 - **How to represent uncertainty (randomness)?**
 - 2. What decides the rewards?**
 - 3. What to be maximized?**
-
- **One possible choice: Markov Decision Processes (MDPs)**

MDPs and Markov Property

- A RL task that satisfies the **Markov property** is called a Markov decision process (MDP).
- **Markov property**: “given the present, the future is independent of the history”.
 - The one-step dynamics is all you need to predict the next state and expected next reward. **Knowing the past changes nothing.**

Markov Property

- In Markov chains:

$$\mathbb{P}(S_{t+1} | S_1, S_2, \dots, S_t) = \mathbb{P}(S_{t+1} | S_t)$$

- In Markov decision processes:

$$\mathbb{P}(S_{t+1} | S_1, \dots, S_t, A_1, \dots, A_t) = \mathbb{P}(S_{t+1} | S_t, A_t)$$

- i.e. given the present, the future is independent of the history

Examples

- **Markovian examples:**
 - Most board games, e.g., Chess, Chinese Chess, Checker.
 - Throw a dice/coin.
- **Non-Markovian examples:**
 - Speed of a moving car.
 - Robot Vacuum cleaner.
- **[Question] Are Evolutionary Algorithms Markovian or non-Markovian?**

Markov Decision Processes (MDPs)

- ❖ Markov chains (basic concepts)
- ❖ **Markov decision processes (MDPs)**
- ❖ Planning in MDPs
- ❖ Extensions to MDPs

Markov Decision Processes (MDPs)

$$M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$$

- \mathcal{S} : set of all possible states.
 - \mathcal{A} : set of all possible actions.
 - $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, transition probability function.
 - \mathcal{R} : immediate reward function.
-
- If both \mathcal{S} and \mathcal{A} are finite, then M is a **finite** MDP.
 - MDPs formally describe an environment for RL where the environment is **fully observable**.

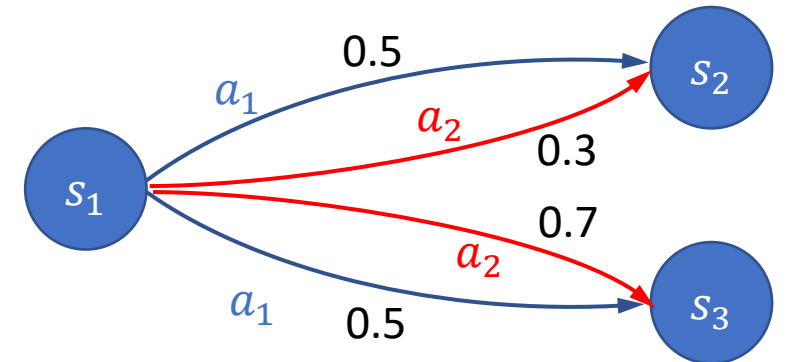
Transition Probability Function

- $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, transition probability function
$$\mathcal{P}(s, a, s') = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

1) Time-independent.

2) $\sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') = 1$.

3) $\sum_{a \in \mathcal{A}, s' \in \mathcal{S}} \mathcal{P}(s, a, s')$ can be greater than 1.



Example: Recycling Robot

- A mobile robot has the job of collecting empty cans in an environment.
- We only consider the high-level decisions about how to search for cans by a RL agent based on **the current charge level of the battery**.
- At each time step, the robot decides whether it should
 - 1) actively **search** for a can for a certain period of time,
 - 2) remain stationary and **wait** for someone to bring it a can, or
 - 3) head back to its home base to **recharge** its battery.
- The state is determined by the state of the battery.
- Considerations (assumptions):
 - The best way to find cans is to actively **search** for them, but this runs down the robot's battery.
 - **Waiting** does not run down the battery but the robot will never find a can.
 - Whenever the robot is searching, the **possibility** exists that its battery will become depleted: In this case the robot must shut down and wait to be rescued (resulting in **negative reward**).

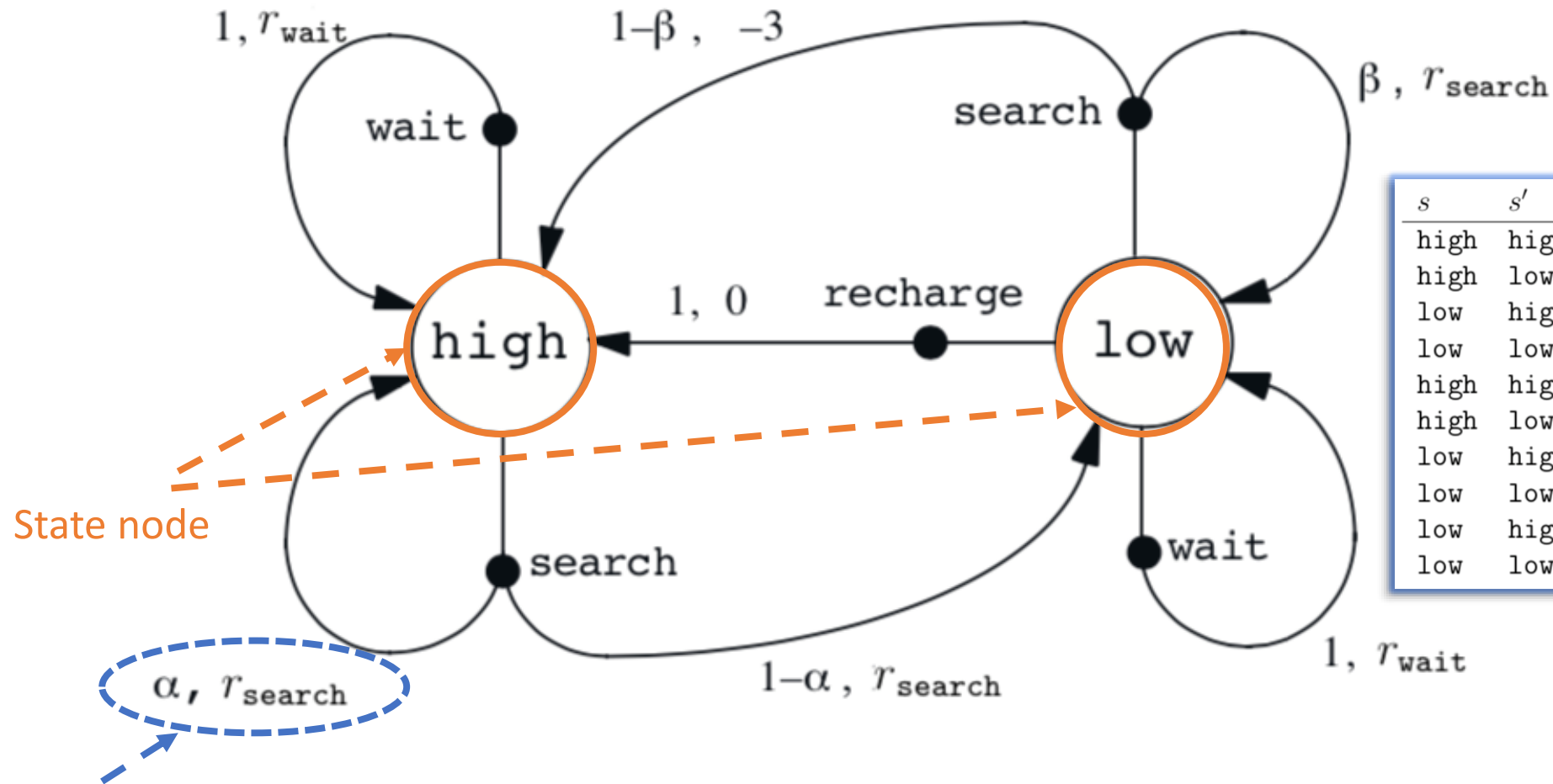
Recycling Robot Example:

Transition Probabilities and Expected Rewards

s	s'	a	$p(s' s, a)$	$r(s, a, s')$
high	high	search	α	r_{search}
high	low	search	$1 - \alpha$	r_{search}
low	high	search	$1 - \beta$	-3
low	low	search	β	r_{search}
high	high	wait	1	r_{wait}
high	low	wait	0	r_{wait}
low	high	wait	0	r_{wait}
low	low	wait	1	r_{wait}
low	high	recharge	1	0
low	low	recharge	0	0.

Transition probabilities and expected rewards for the finite MDP of the recycling robot example [Sutton&Barto].

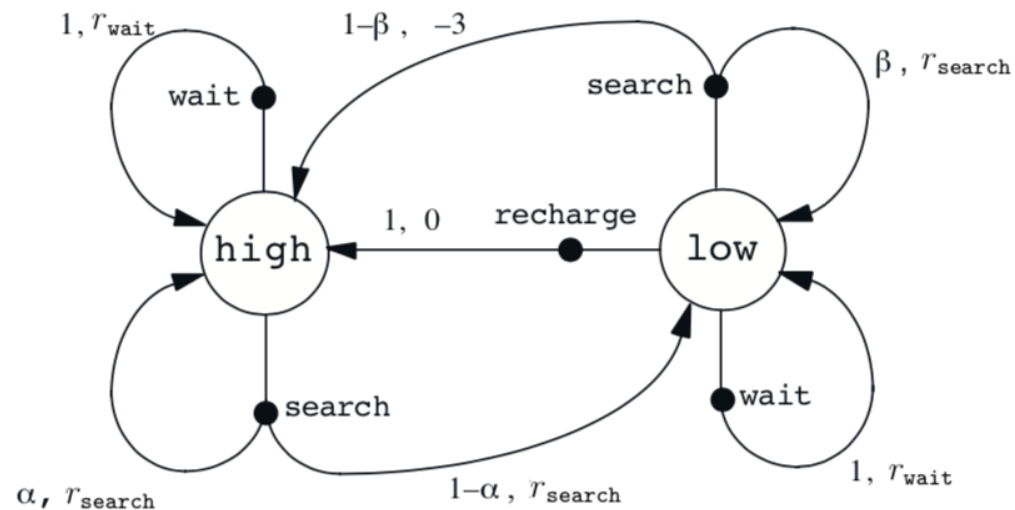
Recycling Robot Example: Transition Graph



s	s'	a	$p(s' s, a)$	$r(s, a, s')$
high	high	search	α	r_{search}
high	low	search	$1 - \alpha$	r_{search}
low	high	search	$1 - \beta$	-3
low	low	search	β	r_{search}
high	high	wait	1	r_{wait}
high	low	wait	0	r_{wait}
low	high	wait	0	r_{wait}
low	low	wait	1	r_{wait}
low	high	recharge	1	0
low	low	recharge	0	0.

Transition graph for the recycling robot example [Sutton&Barto].

Recycling Robot Example: Transition Graph

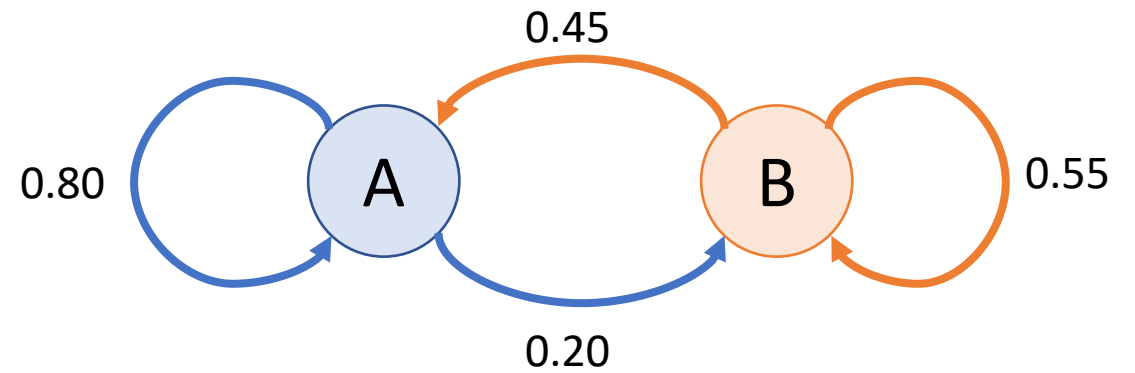


Example: Starting from $S_0 = \text{high}$

- $\text{high} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{search}} \text{low}$
- $\text{high} \xrightarrow{\text{search}} \text{high} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{search}} \text{high}$
- $\text{high} \xrightarrow{\text{search}} \text{high} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{recharge}} \text{high}$
- $\text{high} \xrightarrow{\text{search}} \text{high} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{recharge}} \text{high}$
- $\text{high} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{wait}} \text{low} \xrightarrow{\text{wait}} \text{low}$
- $\text{high} \xrightarrow{\text{wait}} \text{high} \xrightarrow{\text{search}} \text{low} \xrightarrow{\text{search}} \text{high}$

Transition Graph -> Matrix

	A	B
A	$P(A A)=0.80$	$P(B A)=0.20$
B	$P(A B)=0.45$	$P(B B)=0.55$



Core Elements of RL

- **Policy**
- **Reward**
- **Value**
- **Model**

We will explore them one by one.

Core Elements of RL: Policy

- MDP describes the environment and the interaction process.
- A policy defines an agent's behaviour (mapping from state to action), i.e. how the agent acts in the given circumstance.
- A policy can be:
 - A function.
 - A look-up table.
 - A search process., e.g., Monte-Carlo Tree Search (MCTS).

Deterministic vs Stochastic Policy

- A **deterministic** policy π is a mapping from states to actions
 $\pi: \mathcal{S} \mapsto \mathcal{A}$
 - $\pi(s) = a$ instructs the agent to take action a at state s
- A **stochastic** policy π is a conditional probability distribution over actions given states
 - $\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$
- Both are **stationary** (time-independent) and **history-independent**.

What If Not History-Independent?

- **History can be “incorporated” into states**
 - Assume $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ is not Markovian.
 - $\mathcal{P}(h, a, s')$ gives transition probability given interaction history h , current action a and next state s' .
 - Define \mathcal{H} as the set of all possible interaction histories, i.e. sequences of states in \mathcal{S} and actions in \mathcal{A} .
 - e.g. $h_1 = [s_1 a_2 s_3 a_1 s_3 a_1 s_6 a_3 s_2 a_2 s_4]$
 - Define \mathcal{P}^+ as the transition probability given history and current action
 - $\mathcal{P}^+(h, a, h') = \mathcal{P}(h, a, s')$ where $h' = [has']$
 - Then $M^+ = \langle \mathcal{H}, \mathcal{A}, \mathcal{P}^+, \mathcal{R} \rangle$ is an MDP.

Core Elements of RL: Rewards

- \mathcal{R} : (immediate) reward function
- Can be defined as $\mathcal{R}(s)$, $\mathcal{R}(s, a)$, or $\mathcal{R}(s, a, s')$:
 - $\mathcal{R}: \mathcal{S} \mapsto \mathbb{R}$
 - $\mathcal{R}: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$
 - $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$
- $\mathcal{R}(s, a, s')$ is more general than $\mathcal{R}(s)$ and $\mathcal{R}(s, a)$
 - $\mathcal{R}(s) = \mathcal{R}(s, *) = \mathcal{R}(s, *, *)$;
 - $\mathcal{R}(s, a) = \mathcal{R}(s, a, *)$
- $\mathcal{R}(s)$ and $\mathcal{R}(s, a)$ are more convenient for matrix representation.

Core Elements of RL: Value

- A **reward**: a scalar feedback provided by the environment only!
 - Learn from the reward signal.
- A **value function**: a prediction of future reward which is used to **evaluate** state(s) so as to select **hopefully optimal action(s)**.
 - Evaluate states = evaluated the actions led to this state.
 - Not easy to determine values or design good value functions.
 - **Efficiently estimating values is crucial!**
 - E.g., expected **discounted cumulative return/reward**.

Cumulative Reward

- When choosing actions, the agent should try to maximize *total amount of reward* rather than just a single-step reward.
 - This is called “cumulative reward” or “return”.
- Since at time t , nothing can be done to R_1, \dots, R_t , the agent only should consider future rewards R_{t+1}, R_{t+2}, \dots
- A simple sum of all rewards $R_{t+1} + R_{t+2} + \dots + R_{t+k} + \dots$?
 - As $k \rightarrow \infty$, this sum might grow infinitely.

Discounted Cumulative Reward

- Common definition: **discounted cumulative reward**

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

- $\gamma \in (0, 1)$ is a **discount factor/rate**
 - γ close to 1 \rightarrow far-sighted; γ close to 0 \rightarrow myopic.
 - Mathematically equivalent to the setting where the agent suddenly dies (and thus future rewards become unimportant) with probability $1 - \gamma$ at each step
 - Usually pick some value near 1 (e.g. 0.997).
- Corollary: $G_t = R_{t+1} + \gamma G_{t+1}$

n -step Cumulative Reward

- Alternative definition: **n -step cumulative reward**

$$G_t^n = \sum_{k=1}^n R_{t+k}$$

- $\gamma = 1$ but only consider n -steps starting from $t + 1$.
- Advice: only use this definition if such n is directly specified by the application.

Value Functions

- Considering the uncertainty of state transitions, it is more reliable to maximize the **expected cumulative reward** $\mathbb{E}_{\pi}[G_t]$ instead.
- E.g.: **expected discounted cumulative reward** with discounted factor $\gamma \in [0, 1]$ (long-/short-term desirability of states):
$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t] = \mathbb{E}_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

Value Functions

- Define state value function $v_\pi: \mathcal{S} \mapsto \mathbb{R}$ of policy π as

$$v_\pi(s) = \mathbb{E}[G_t | \pi, S_t = s]$$

- Define action value function $q_\pi: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ of policy π as

$$q_\pi(s, a) = \mathbb{E}[G_t | \pi, S_t = s, A_t = a]$$

Bellman Equation for $v_\pi(s)$

Recall: $v_\pi(s) = \mathbb{E}_\pi[G_t] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t | \pi, S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | \pi, S_t = s] \\ &= \mathbb{E}[\boxed{R_{t+1}} + \boxed{\gamma v_\pi(S_{t+1})} | \pi, S_t = s] \end{aligned}$$

Discounted value of expected next state

For deterministic $\pi(s)$, since $\mathbb{P}(S_{t+1} = s' | S_t = s) = \mathcal{P}(s, \pi(s), s')$, it becomes

$$v_\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, \pi(s), s') (\mathcal{R}(s, \pi(s), s') + \gamma v_\pi(s')).$$

Bellman Equation for $v_{\pi}(s)$

For stochastic $\pi(a|s)$, it becomes

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') (\mathcal{R}(s, a, s') + \gamma v_{\pi}(s'))$$

- The equation for $\pi(s)$ is a special case of the one for $\pi(a|s)$.
- However, using $\pi(s)$ is sufficient in many cases (will be discussed later).

Matrix Representation

- When the reward function is $\mathcal{R}(s, a)$, we have:
 - $\mathcal{R}_\pi(s) = \sum_a \pi(a|s) \mathcal{R}(s, a)$ or $\mathcal{R}(s, \pi(s))$.
 - So \mathcal{R}_π can be written as $|\mathcal{S}| \times 1$ vector.
 - \mathcal{P}_π : $|\mathcal{S}| \times |\mathcal{S}|$ matrix.
 - v_π : $|\mathcal{S}| \times 1$ vector.
- Then the Bellman equation becomes $v_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi v_\pi$
- Then $\mathcal{R}_\pi = (I - \gamma \mathcal{P}_\pi) v_\pi$
- Then $v_\pi = (I - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi$
 - thus can be solved directly.

Bellman Equation for $q_\pi(s, a)$

Recap: $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s, A_t = a]$

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[G_t | \pi, S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | \pi, S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | \pi, S_t = s, A_t = a] \end{aligned}$$

Since action at the current step is always a regardless of π in $q_\pi(s, a)$,

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') (\mathcal{R}(s, a, s') + \gamma v_\pi(s')).$$

Bellman Equation for $q_{\pi}(s, a)$

Compare

$$\mathbf{v}_{\pi}(\mathbf{s}) = \left[\sum_{a \in \mathcal{A}} \pi(a|\mathbf{s}) \right] \sum_{s' \in \mathcal{S}} \mathcal{P}(\mathbf{s}, a, s') (\mathcal{R}(\mathbf{s}, a, s') + \gamma \mathbf{v}_{\pi}(s'))$$

and

$$\mathbf{q}_{\pi}(\mathbf{s}, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(\mathbf{s}, a, s') (\mathcal{R}(\mathbf{s}, a, s') + \gamma \mathbf{v}_{\pi}(s'))$$

we have

$$\mathbf{v}_{\pi}(\mathbf{s}) = \sum_{a \in \mathcal{A}} \pi(a|\mathbf{s}) \mathbf{q}_{\pi}(\mathbf{s}, a)$$

Bellman Equation for $q_\pi(s, a)$

By $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$, we have:

$$\begin{aligned} q_\pi(s, a) &= \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') (\mathcal{R}(s, a, s') + \gamma v_\pi(s')) \\ &= \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') (\mathcal{R}(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')). \end{aligned}$$

For deterministic π ,

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') (\mathcal{R}(s, a, s') + \gamma q_\pi(s', \pi(s'))).$$

Markov Decision Processes (MDPs)

- ❖ Markov chains (basic concepts)
- ❖ Markov decision processes (MDPs)
- ❖ **Planning in MDPs**
- ❖ Extensions to MDPs

Optimality

- The **optimal state-value function** is defined as $\forall s \in \mathcal{S}$

$$v^*(s) = \max_{\pi} v_{\pi}(s)$$

- The **optimal action-value function** is defined as $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$

$$q^*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- If a policy π satisfies $v_{\pi}(s) = v^*(s)$ and $q_{\pi}(s, a) = q^*(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, then it is called an **optimal policy** and denoted as π^* .

Optimality

- By definition, optimal policies have the highest expected cumulative reward **at any state**.
- Therefore, the ultimate objective of RL is to find out such optimal policies.
- It can be proved that
$$\forall s \in \mathcal{S}, (v_{\pi}(s) = v^*(s)) \leftrightarrow \forall (s, a) \in \mathcal{S} \times \mathcal{A}, (q_{\pi}(s, a) = q^*(s, a)).$$
- Thus only need to consider one of them.

Optimality

- For any MDP there exists **at least one optimal policy**.
 - An obvious solution is $\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} q^*(s, a)$.
 - Since this is a deterministic policy and it always exists, using stochastic policy $\pi(a|s)$ can be unnecessary if we are only interested in optimal policies.
- In some MDPs, there can be **more than one optimal policy**.
 - Trivial case: if all immediate rewards equal to 0, then all possible policies are optimal.

Bellman Optimality Equations

- $q_{\pi}(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \left(\mathcal{R}(s, a, s') + \gamma q_{\pi}(s', \pi(s')) \right)$
- $\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} q^*(s, a)$

- Therefore,

$$q^*(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \left(\mathcal{R}(s, a, s') + \gamma \max_{a' \in \mathcal{A}} q^*(s', a') \right)$$

- Similarly,

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \left(\mathcal{R}(s, a, s') + \gamma v^*(s') \right)$$

Solving Bellman Equations

- **Given any MDP, its optimal policies can be discovered by solving the Bellman optimality equations.**
- **Since v^* and q^* involve max operator (non-linear), they can't be solved by linear algebra.**
- **Many iterative solution methods:**
 - 1) Policy Iteration;
 - 2) Value Iteration;
 - 3) Q-learning;
 - 4) Sarsa (State-Action-Reward-State-Action).

Iterative Policy *Evaluation*

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) \leftarrow Converge test
Output $V \approx v_\pi$

Figure 4.1 of [Sutton&Barto].

1) Policy *Iteration* (PI)

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)

Policy Evaluation

- *Estimate v_π*

3. Policy Improvement

policy-stable \leftarrow *true*

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

$a \leftarrow \pi(s)$

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

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

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

Policy Improvement

- *Generate better π*

Figure 4.3 of [Sutton&Barto].

1) Policy Iteration (PI)

- In finite MDPs, the policy iteration algorithm is guaranteed to converge to π^* in a finite number of iterations.
- Intuition: whenever the current policy is updated, its values will *never* decrease, and at least one value will increase.

2) Value Iteration (VI) Algorithm

- **Steps:**

- 1) **Start from arbitrary v (can be wrong values like all 0).**

- 2) **Update all $v(s)$ by**

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') (\mathcal{R}(s, a, s') + \gamma v(s'))$$

- 3) **Go to (2) until v converges.**

- **Can be done similarly using q**

2) Value Iteration (VI) Algorithm

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

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

Figure 4.5 of [Sutton&Barto].

Solving Bellman Equations

- In finite MDPs, the value iteration algorithm converge to v^* or q^* as $\#iteration \rightarrow \infty$.
- Value iteration is easier to implement and is usually slightly faster than policy iteration.

Markov Decision Processes (MDPs)

- ❖ Markov chains (basic concepts)
- ❖ Markov decision processes (MDPs)
- ❖ Planning in MDPs
- ❖ **Extensions to MDPs**

Extension to MDPs

- **Stochastic immediate rewards**

- **Original $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is deterministic.**
- **It can also be stochastic $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \mapsto [0, 1]$, i.e. given an arbitrary transition (s, a, s') , the probability of the reward being r is**

$$\mathbb{P}(R_t = r | S_t = s, A_t = a, S_{t+1} = s').$$

- **To compute the expected cumulative reward, we need to use $\mathbb{E}[\mathcal{R}(s, a, s')]$ instead of $\mathcal{R}(s, a, s')$ in Bellman equations.**
- **$\mathbb{E}[\mathcal{R}(s, a, s')]$ can be estimated easily with sample means.**

Extension to MDPs

- **MDPs are fully observable.**
 - All information about the current state is provided to the agent.
- **Many real-world applications are only partially observable.**
 - The agent only receives part of the state information.
 - **Partially Observable Markov Decision Processes (POMDPs)**

POMDPs

- A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{W} \rangle$
 - $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}$: the same as those in MDPs.
 - \mathcal{O} : set of all possible observations.
 - $\mathcal{W}: \mathcal{S} \times \mathcal{O} \mapsto [0, 1]$ is an observation probability function
$$\mathcal{W}(s, o) = \mathbb{P}(\mathbf{O}_t = o | \mathbf{S}_t = s)$$
 - The agent may observe the same o at different states s and s' .
 - It may also observe different o and o' at the same s .

Examples of POMDPs

- **Card games: same observation, different actual states.**
- **Mahjong.**



Examples of POMDPs

- **Inaccurate sensors in robotics: the same actual state, but different observations.**

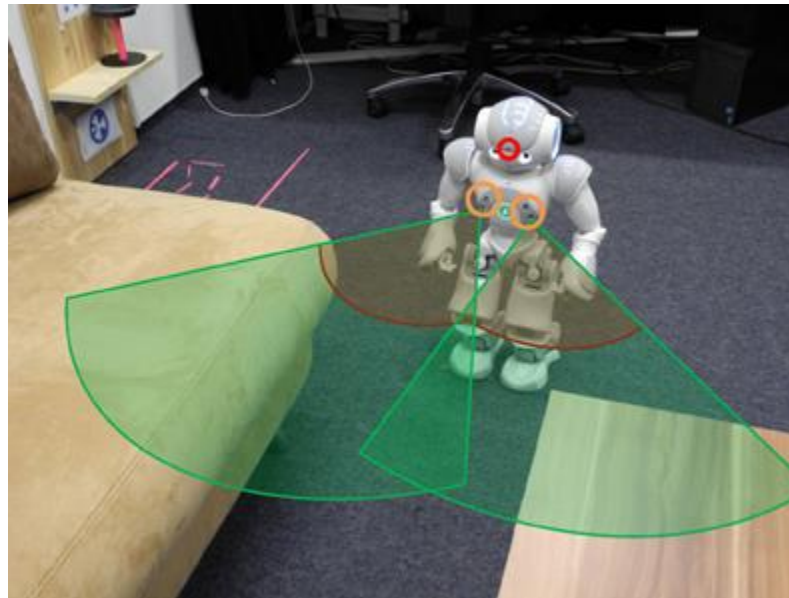


image from <http://journal.frontiersin.org/article/10.3389/fnbot.2013.00015/full>

Dealing with POMDPs

- Simply treat them as regular MDPs
- Maintain a *belief* vector of states
 $\langle \mathbb{P}(S_t = s_1 | H_t), \mathbb{P}(S_t = s_2 | H_t), \dots, \mathbb{P}(S_t = s_n | H_t) \rangle$
conditioned by the observed history H_t
 - Such a vector is called a *belief state*
 - Belief states satisfy the Markov property
- Heuristics/search based methods
 - Bypass the difficulty of value evaluation in POMDPs

Other Extensions

- **Continuous MDPs**
 - Continuous @ state space / action space / time.
 - Function approximation is usually applied.
- **Semi-MDPs**
 - Actions can last for several steps.
 - Transition $\mathcal{P}(s, a, n, s') = \mathbb{P}(S_{t+n} = s' | S_t = s, A_t = a)$.
- **Decentralised POMDPs**
 - Multi-agent: a team of agents cooperate to maximize a global cumulative reward.
 - PO: members only have access to local information.
 - Decentralised: no global leader/planner/commander.

Reading Materials for This Lectures

- [Sutton&Barto] Sutton RS, Barto AG. Reinforcement learning: An introduction. Cambridge: MIT press; 1998 Mar 1.
 - Pages 53-107.
 - <http://incompleteideas.net/sutton/book/the-book-2nd.html>
- Szepesvári C. Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning. 2010 Jul 7;4(1):1-03.
 - <https://sites.ualberta.ca/~szepesva/RLBook.html>
- Wiering M, Van Otterlo M. Reinforcement learning. Springer, 2012.
- David Silver's slides (2015):
<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>