Markov Decision Processes

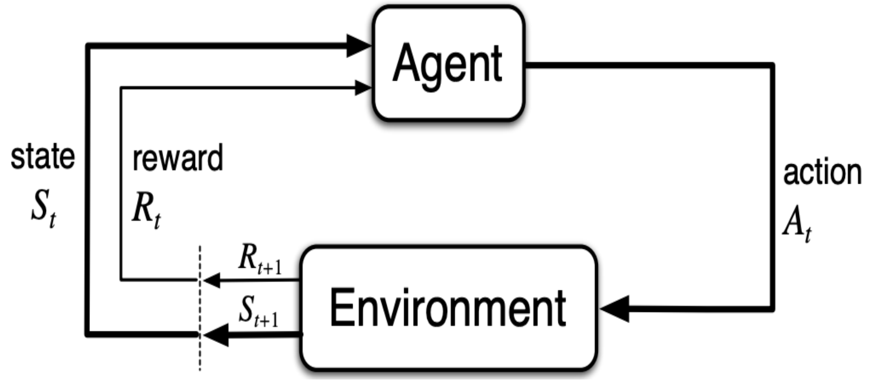
David I. Inouye

PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

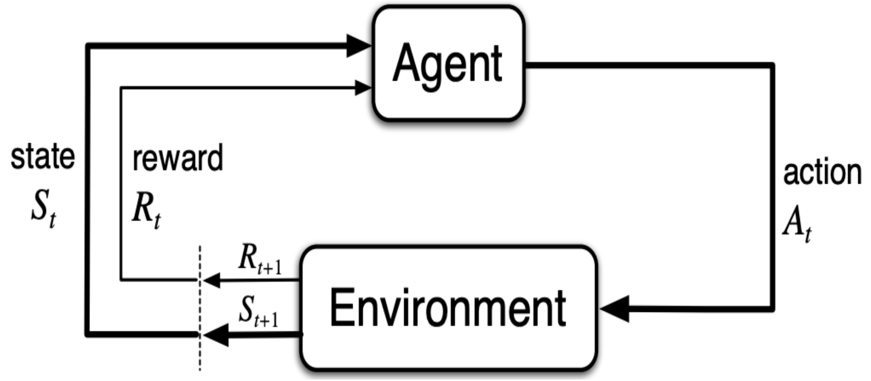
The agent and environment are the two key actors in RL – How should they be defined?

- Example: Humanoid robot
- Is the "agent" the whole robot?
- What "actions" can the "agent" take?
  - Walk forward?
  - Increase power to left leg motor?
- The agent is probably the "controller" of the robot
  - In fact, the position of arms and legs might be part of the "environment" (e.g., they could be stuck or lose power)
- Suppose a leg breaks, the controller cannot fix the leg by itself so the leg is actually a part of the environment
  - The agent could put more power to the other leg and try to balance on one leg but it does not have direct control of the leg



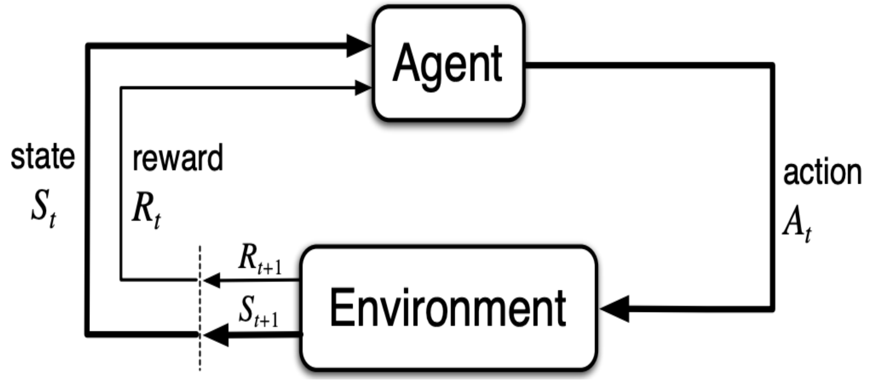
The agent and environment are the two key actors in RL – How should they be defined?

- Example: Oil refinery
- Is the "agent" the whole refinery?
- What "actions" can the "agent" take?
    - Produce x amount of crude oil?
    - Change temperature of this module?
    - Increase the electricity going to fan or heater?
- The agent is again just the "controller" part of the refinery
    - Each module may have its own controller/agent
    - The top-level controller can only "ask" for something (i.e., a command) but it may not happen (e.g., something breaks or impossible to fulfill)



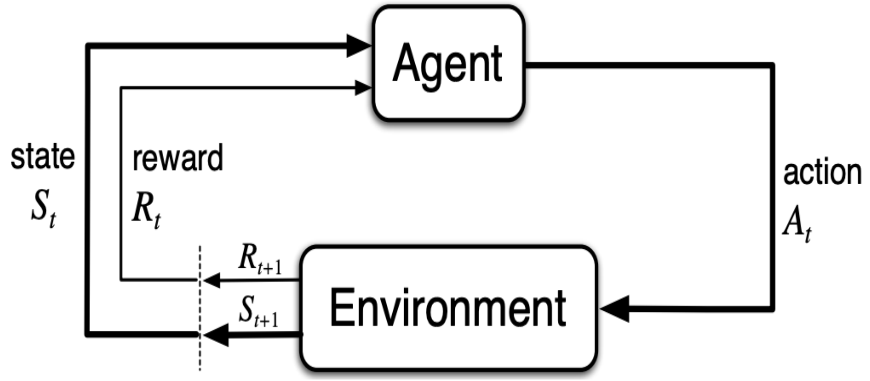
The agent and environment are the two key actors in RL – How should they be defined?

- The <u>**agent**</u> should only be defined as the part that can be **directly and explicitly** controlled via concrete actions
    - Actions – Select how much power send to each robotic motor
    - Actions – Change fan speed
    - Actions – Request subcontrollers to produce a certain amount
    - Note: The boundary should be defined by what can be ***controlled*** rather than what is ***known***
- The <u>**environment**</u> is anything that is NOT the agent (I'd call it the **non-agent** more precisely)
    - Actual position of the robot body
    - Actual temperature in boiler



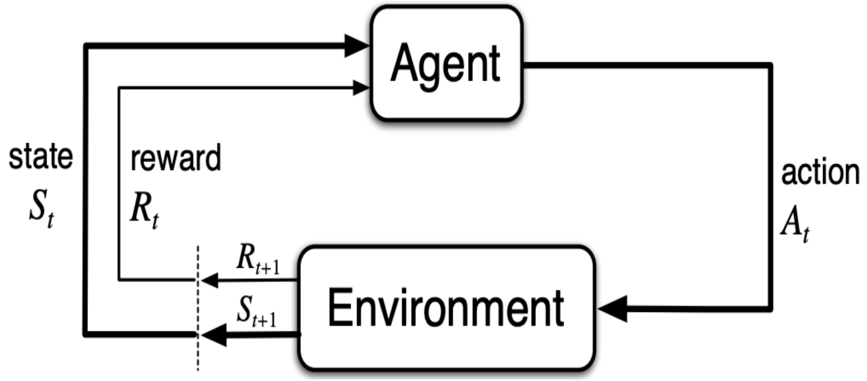
Markov Decision Processes (MDP) mathematically formalize RL problems via random variables

- Sensation / Observation
  - Observing the environment
  - Encoded as a random variable called **state** S_t
- Action
  - Agent's ability to interact with the environment.
  - Encoded as a random variable called **action** A_t
- Reward
  - Immediate feedback from the environment
  - Encoded as a random variable called **reward** R_t



MDPs model a **discrete** sequence of states, actions, and rewards

- At each discrete timestep t
  - Agent receives environment state
    $$S_t \in \mathcal{S}$$
  - Agent decides action based on state,
    $$A_t(S_t) \in \mathcal{A}(S_t)$$
  - Agent receives reward and new state
    $$R_{t+1} \in \mathbb{R}, \qquad S_{t+1} \in \mathcal{S}$$
- This creates a sequence of state, action, reward:
  $$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$



MDPs model the joint distribution of this sequence by **assuming conditional independence**

- Using the chain rule, we could define the probability of the following sequence $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$

  $p(S_0)p(A_0|S_0)p(S_1, R_1|S_0, A_0)p(A_1|S_0, A_0, S_1, R_1)p(S_2, R_2|S_0, A_0, S_1, R_1, A_1) \dots$
- MDPs assume conditional independence
  - **Action** only depends on **current state**
  - **New state and reward** only depend on **current state and action** (Markov part of MDP)

$$p(S_0)p(A_0|S_0)p(S_1, R_1|S_0, A_0)p(A_1|S_0, \cancel{A_0, S_1, R_1})p(S_2, R_2|\cancel{S_0, A_0,} S_1, \cancel{R_1,} A_1) \dots$$

- The **environment's** dynamics are completely represented by:

$$p(S_t, R_t|S_{t-1}, A_{t-1})$$

- The **agent's** dynamics are completely represented by a policy:

$$\pi(A_t|S_t) \coloneqq p(A_t|S_t)$$

The goal or ultimate objective is to find a policy that maximizes the (weighted) sum of **future** rewards

- While rewards are immediate, the goal is to maximize the long-term **expected return**, i.e., sum of future rewards
- **Episodic tasks** have a special **terminal state** and then are reset to a standard starting state
  - Usually assumed to be completely independent
  - Examples: Game playing or trips through a maze
  - Expected return could just be a sum of **future** rewards: $G_t = \sum_{k=0}^{T} R_{t+k+1}$
- **Continual tasks** do not have a clear terminal state
  - A simple sum of rewards would not converge (i.e., infinite)
  - Examples: Controlling an oil refinery or a robot learning to walk
  - Expected return is usually a **weighted** sum of future rewards where γ is the **discount rate**

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad \text{where } 0 < \gamma < 1$$

- If we assume that for episodic tasks, $\forall t > T, R_t = 0$, we can simply use the notation for continual tasks but allow $\gamma_t = 1$ if it is episodic

Returns at the current timestep are equal to rewards plus discounted returns at next timestep

- $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- $= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$
- $= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots)$
- $= R_{t+1} + \gamma G_{t+1}$

- Thus, returns are recursively related to each other
    - This simple relationship is fundamental to theoretic development later

The **state-value function** defines the expected return of a state ***given*** a particular policy π

- Remember, rewards are easy. Estimating value is hard.
- Estimating value is the key to most RL algorithms.
- First, we define the value of **states**.
- The **state-value function** for policy π is

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \end{aligned}$$

  - The expectation is based on the given policy π (e.g., random/greedy policy)
  - How "good" is this state, ***given*** that the agent follows π for all actions afterwards?
  - **Important:** This state-value function will be different for different policies.

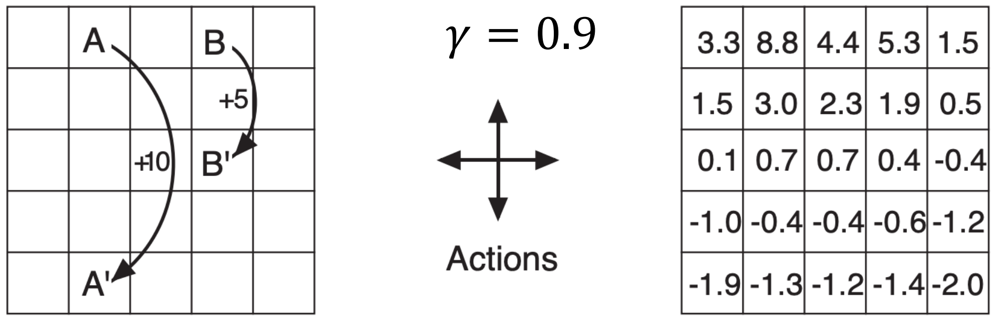
The **Bellman equation** for v_π is a special recursive equation whose solution is the value function

- $v_\pi(s) \coloneqq \mathbb{E}_\pi[G_t | S_t = s]$
- $= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s]$, (see previous slide)
- $= \sum_{a,r,s',a',r',s'' \dots} \pi(a|s)\, p(s',r|s,a) \pi(a'|s') p(s'',r'|s',a') \dots [\dots]$
- $= \sum_{a,r,s'} \pi(a|s)\, p(s',r|s,a) \left[r + \gamma \sum_{a',r',s'',\dots} \pi(a'|s') p(s'',r'|s',a') G_{t+1}\right]$
- $= \sum_{a,r,s'} \pi(a|s)\, p(s',r|s,a) \big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']\big]$
- $= \sum_{a,r,s'} \pi(a|s)\, p(s',r|s,a) [r + \gamma v_\pi(s')]$

- $v_\pi(s) = \sum_{a,r,s'} \pi(a|s)\, p(s',r|s,a) [r + \gamma v_\pi(s')]$

Example: Solved Bellman equation (which is a system of linear equations) to find the value function for a random policy

Example 3.5: Gridworld Figure 3.2 (left) shows a rectangular gridworld representation of a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: `north`, `south`, `east`, and `west`, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of $+10$ and take the agent to A$'$. From state B, all actions yield a reward of $+5$ and take the agent to B$'$.



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

Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

- $v_\pi(s) = \sum_{a,r,s'} \pi(a|s)\, p(s',r|s,a)[r + \gamma v_\pi(s')]$
- $v_\pi([1,1]) = N + S + E + W$
- $= \frac{1}{4}\Big(\big(-1 + \gamma(3.3)\big) + \big(0 + \gamma(1.5)\big) + \big(0 + \gamma(8.8)\big) + \big(-1 + \gamma(3.3)\big)\Big)$
- $= \frac{1}{4}\Big(\big(-1 + 0.9(3.3)\big) + \big(0 + 0.9(1.5)\big) + \big(0 + 0.9(8.8)\big) + \big(-1 + 0.9(3.3)\big)\Big)$
- $= 3.3 = v_\pi([1,1])$

Example copied verbatim from page 60 of Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

The **action-value function** defines the expected return of taking an action ***given*** a current state AND a particular policy π

- We now define the value of an **action** given a current state and policy π
- The **action-value function** for policy π is defined as:

$$q_\pi(s,a) \coloneqq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

  - The key difference is that now we condition on the next action a
  - The expectation is based on the given policy π (e.g., random/greedy policy)
  - How "good" is the **action**, ***given*** that the agent first takes action a and then follows π for all actions afterwards?
  - **Important:** This action-value function will be different for different policies.

Optimal policies and optimal value functions can be defined and exist theoretically

- How should we define an optimal policy?
    - We want it to mean that this policy will produce the most expected return (i.e., long-term reward).
- One policy π is better than another policy π' if it's value function is better for all possible states

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s), \qquad \forall s \in \mathcal{S}$$

- There exists at least one policy that is better or equal to all others called an optimal policy

$$\pi_* \geq \pi, \qquad \forall \pi$$

- All optimal policies (possibly more than one) share the same **optimal state-value function**

$$v_*(s) \coloneqq \max_\pi v_\pi(s), \qquad \forall s \in \mathcal{S}$$

- Similarly, they share the same **optimal action-value function**

$$q_*(s,a) = \max_\pi q_\pi(s,a), \qquad \forall s \in \mathcal{S}$$

The **Bellman optimality** equation for v_* enables solving for the **optimal** state-value function

- $v_*(s) = \max_a q_{\pi_*}(s, a)$
- $= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a]$
- $= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a]$
- $= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$
- $= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')]$

- No optimal policy here! Just the model part since we are taking maximum over actions.
- For finite MDPs, this can be solved via a system of non-linear equations

The **Bellman optimality equation** for q_* enables solving for the **optimal** action-value function

- $q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q_*(s,a) \,\middle|\, S_t = s, A_t = a\right]$
- $= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$

- Again, no policy here, just the q function!
- This can be solved as well.

The optimal value functions can be used to construct an **optimal policy**!

- If we have $v_*(s)$, then we can simply choose the optimal action that will maximize the value after taking one action

$$a_t^* = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

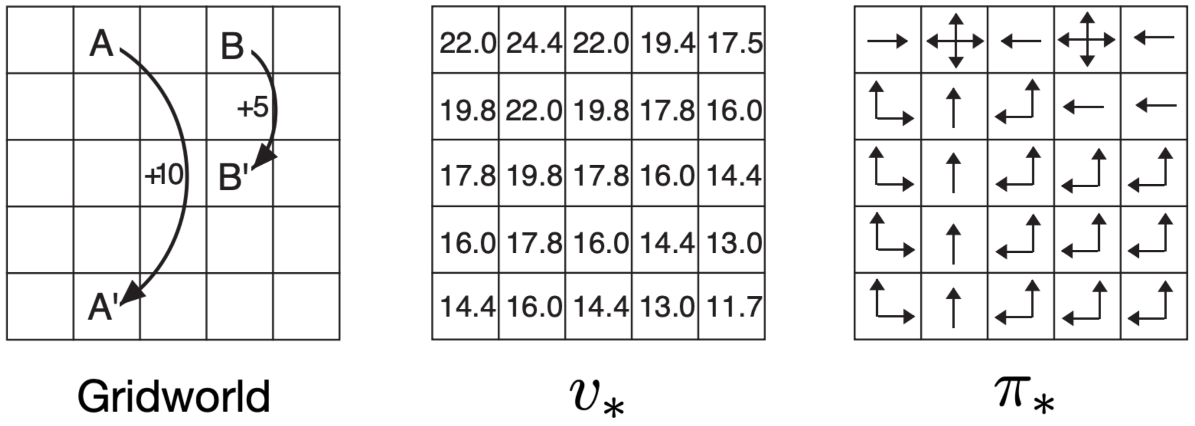
  - Called one-step search or **greedy search** *with respect to state-value function* $v_*(s)$ (rather than greedy *with respect to* the expected immediate reward $\mathbb{E}[R_t|A_t = a]$)
  - **Greedy/local** w.r.t. v_* is **globally** optimal.
- If we have $q_*(s,a)$, then the optimal action is even simpler:

$$a_t^* = \arg\max_a q_*(s,a)$$

  - The action-value function "caches" the one-step search values

Example: With the Gridworld optimal value function, we can define the optimal policy

Example 3.8: Solving the Gridworld Suppose we solve the Bellman equation for v_* for the simple grid task introduced in Example 3.5 and shown again in Figure 3.5 (left). Recall that state A is followed by a reward of +10 and transition to state A′, while state B is followed by a reward of +5 and transition to state B′. Figure 3.5 (middle) shows the optimal value function, and Figure 3.5 (right) shows the corresponding optimal policies. Where there are multiple arrows in a cell, all of the corresponding actions are optimal.



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

Figure 3.5: Optimal solutions to the gridworld example. ■

MDPs solved!? No…
Solution relies on several difficult assumptions

1. The dynamics of the environment are known (i.e., $p(S_t, R_t | S_{t-1}, A_{t-1})$ is known perfectly)
    - Except in simple simulations like grid world, these dynamics are rarely known
    - Even for simple games like chess, the opponent's strategy is unknown so the dynamics are unknown
2. Computational resources are sufficient for this calculation
    - For example, even a simple backgammon game has 10^{20} states
3. The states have the Markov property
    - All relevant information of the past must represented in the environment state
    - It is often difficult to ensure this property in real-world scenarios

RL is about approximating solutions to these MDPs

- While optimal solutions are almost never possible, approximations can still be quite useful
- Most RL algorithms focus on estimating the value functions in some way
- RL algorithms often substitute knowledge of the environment with actual experience in place of knowledge (like in bandits)
- One saving grace in practice is that not all states are equally likely
  - This is akin to the notion that real-world data (e.g., images) is much simpler than the high-dimensional space it lives in
  - Thus, generalization beyond the theoretic "worst case" is possible in practice
- While there are other algorithms for solving MDPs, the **online nature** of RL distinguishes it from other approaches to MDPs
  - It can collect more information on the "common" states and thus do well

Summary of MDPs

- Agent and environment mappings for problems
  - The agent is defined as what is **directly controllable** (NOT mere knowledge)
  - The environment is everything else
- MDPs model this interaction with **states**, **actions** and **rewards**
  - The environment is defined by the transition distribution $p(S_t, R_t | S_{t-1}, A_{t-1})$
  - The agent is defined by a policy $\pi(A_t | S_t)$
  - Together, they define a joint distribution over sequences of states, actions and rewards
- The agent attempts to optimize the **expected return**, which is the discounted sum of future rewards
- The **state-value and action-value functions** represent the long-term value of states or actions
- An **optimal policy** can be constructed from optimal value functions
- In practice, RL algorithms usually approximate these value functions in an **online manner** even if the environment is unknown

Reference

- Based on the excellent RL book by Sutton and Barto
    - http://incompleteideas.net/book/the-book-2nd.html