Reinforcement Learning - 2

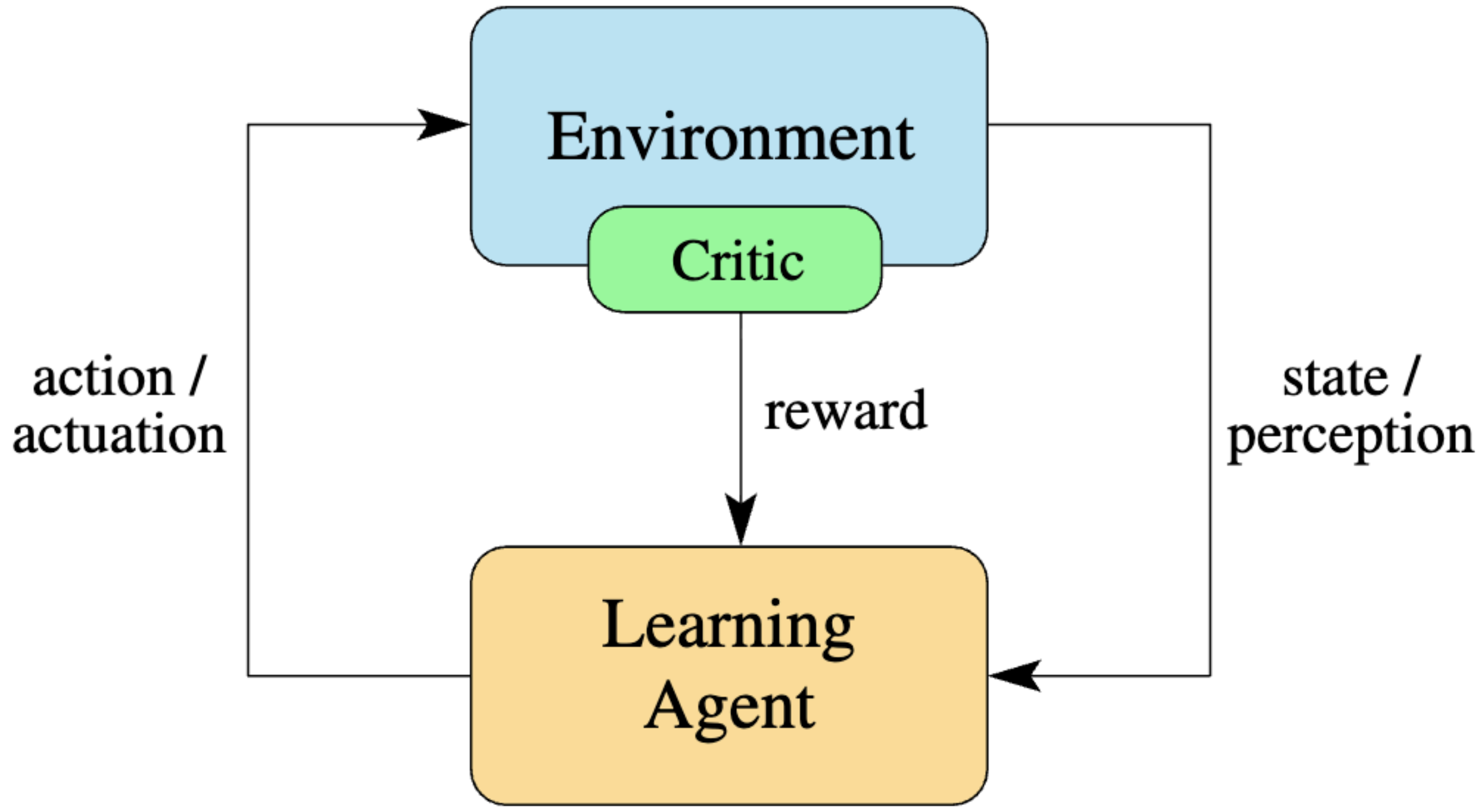
Recap

- **Last class.**
  - Hand-wavy introduction to reinforcement learning
    - How it differs from supervised learning
  - Formalisms
    - Markov chain
    - Policy

- **Today.**
  - Markov Process / MRP / MDP
  - Bellman's equation

Recap: RL Framework

- Learning to solve sequential decision-making, without supervision
  - Can break down complicated problems into easier sub-problems
  - Can adapt to any change in environment
  - Easier to generalize
- **Goal.** Select actions to maximize the total expected future reward



for $t = 1, \ldots, n$ **do**
    The agent perceives state s_t
    The agent performs action a_t
    The environment evolves to s_{t+1}
    The agent receives reward r_t
end for

Markov Process

- Let t be the time clock
- A **Markov process** is a tuple (S, p)
  - S: state space
  - p: state transition probability
- We make a Markov assumption on p
  - The next state s_{i+1} is determined solely by the current state s_i,

$$p(s_{t+1} \mid s_1, \ldots, s_t) = p(s_{t+1} \mid s_t)$$

  - That is, "state" is rich enough to contain all relevant information
- Note. No rewards, no actions

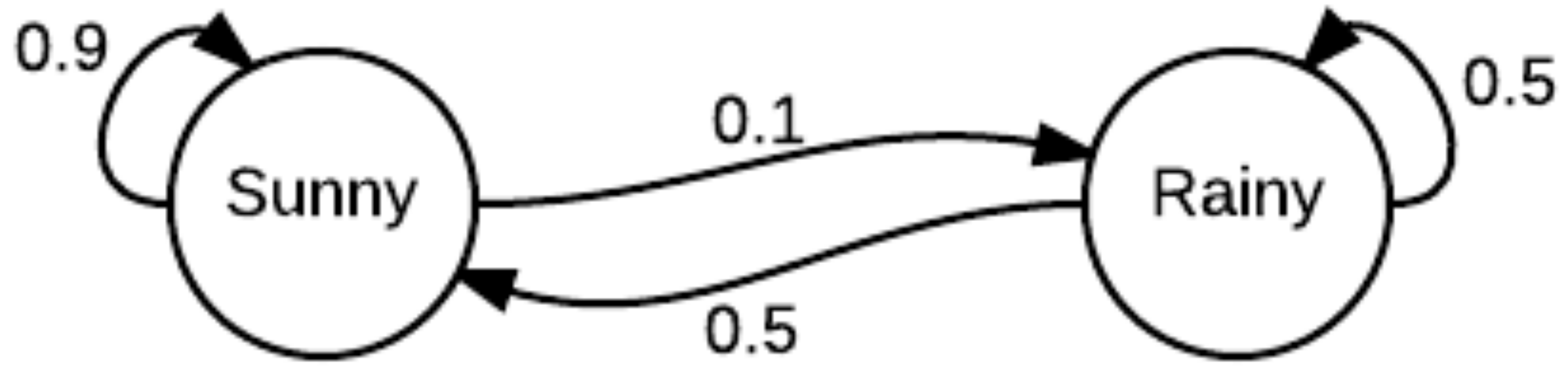
Markov Process

- If the number of states is finite, the transition dynamics can be expressed via state transition matrix
  - If states are $S = \{z_1, \cdots, z_N\}$, then we'll have

$$P = \begin{bmatrix} p(z_1 | z_1) & P(z_2 | z_1) & \cdots & p(z_N | z_1) \\ p(z_1 | z_2) & P(z_2 | z_2) & \cdots & p(z_N | z_2) \\ & & \cdots & \\ p(z_1 | z_N) & P(z_2 | z_N) & \cdots & p(z_N | z_N) \end{bmatrix}$$

- **Example.** Weather transition model

$$\begin{bmatrix} p_{\text{sunny}}^{(t+1)} \\ p_{\text{rainy}}^{(t+1)} \end{bmatrix} = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \begin{bmatrix} p_{\text{sunny}}^{(t)} \\ p_{\text{rainy}}^{(t)} \end{bmatrix}$$



Markov Reward Process

- A **Markov reward process (MRP)** is a tuple (S, p, r)
  - a Markov process (S, p)
  - a reward r
- The reward is earned by transitioning from one state to another
  - Deterministic: $r_t = r(s_t, s_{t+1})$
  - Stochastic: $r_t \sim r(\cdot \,|\, s_t, s_{t+1})$
- For simplicity, assume that it depends on the current state only
  - We can define the **reward function**

$$R(s) = \mathbb{E}[r_t \,|\, s_t = s]$$

Markov Reward Process

- We are interested in the **expected accumulated rewards** (or "return")

$$G_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{H-1} r_{t+H-1}$$

  - γ: discount factor
  - H: "horizon," i.e., the number of steps in each episode
    - can be infinite

- The **state value function** is the expected return, starting at state s

$$\begin{aligned} V(s) &= \mathbb{E}[G_t \,|\, s_t = s] \\ &= \mathbb{E}[G_t \,|\, r_t + \gamma r_{t+1} + \cdots + \gamma^{H-1} r_{t+H-1} \mid s_t = s] \end{aligned}$$

Markov Reward Process

- If we have an infinite horizon, then we have:

$$V(s) = R(s) + \gamma \sum_{s' \in S} p(s' \mid s) V(s)$$

  - Value = immediate reward + discounted sum of future rewards
  - Note. This is called the Bellman equation (for MRPs)
- For a finite-state MRP, we have a neat matrix form

$$\begin{bmatrix} V(s_1) \\ \cdots \\ V(s_N) \end{bmatrix} = \begin{bmatrix} R(s_1) \\ \cdots \\ R(s_N) \end{bmatrix} + \gamma \begin{bmatrix} p(s_1 \mid s_1) & \cdots & p(s_N \mid s_1) \\ & \cdots & \\ p(s_1 \mid s_N) & \cdots & p(s_N \mid s_N) \end{bmatrix} \begin{bmatrix} V(s_1) \\ \cdots \\ V(s_N) \end{bmatrix}$$

  - More simply, we can write

$$V = R + \gamma P V$$

Markov Reward Process

$$V = R + \gamma PV$$

- Solving the matrix equation, we have

$$V = (I - \gamma P)^{-1} R$$

  - Fortunately, the matrix $(I - \gamma P)$ is invertible
- **Computation.**
  - Direct inverse: Requires $O(N^3)$
  - Iterative solution: Requires $O(N^2)$ per step
    - Initialize $V_{(0)} = \mathbf{0}$
    - Repeat $V_{(k+1)} = R + \gamma PV_{(k)}$ ← Converges, theoretically (Fixed point theorem)

Markov Decision Process

- A **Markov decision process (MRP)** is a tuple (S, A, p, r)
  - an action space A
  - a Markov reward process (S, p, r), with
    - Action-dependent transition model

$$p(s_{t+1} \mid s_t, a_t)$$

    - Action-dependent reward

$$r_t \sim r(\,\cdot \mid s_t, s_{t+1}, a_t)$$

      - Again, for simplicity, assume independence w.r.t. s_{t+1}

$$R(s, a) = \mathbb{E}[r_t \mid s_t = s, a_t = a]$$

MDP Policies

- Agent's behaviors are specified by policies
  - Deterministic: $a_t = \pi(s_t)$
  - Stochastic: $a_t \sim \pi(\cdot \mid s_t)$

- If we fix the policy π, then MDP becomes a MRP
  - The return function and the state transitions are given as

$$R^\pi(s) = \sum_{a \in A} \pi(a \mid s) R(s, a)$$

$$p^\pi(s' \mid s) = \sum_{a \in A} \pi(a \mid s) p(s' \mid s, a)$$

MDP Policies

- Now, we can compute the value function $V(s)$ for this MRP
  - Similarly to MRP, we have a Bellman equation

$$V^{\pi}(s) = \sum_{a \in A} \pi(a \mid s)\left(R(a, s) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^{\pi}(s)\right)$$

- Again, the solution can be found by an iterative method:
  - Initialize: $V^{\pi}_{(0)}(s) = \mathbf{0}$
  - Iterate:

$$V^{\pi}_{(k+1)}(s) = \sum_{a \in A} \pi(a \mid s)\left(R(s, a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^{\pi}_{(k)}(s')\right)$$

Optimal Policy for MDP

- Given the initial state s, the expected return of a policy π is: $V^{\pi}(s)$
  - We are interested in the optimal policy:

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

- Mathematically, we can show that:
  - Exists a solution
  - The solution is unique
- For an infinite-horizon problem, the solution is:
  - Deterministic
  - Stationary

Finding the optimal policy

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

- **Question.** How do we solve the optimization?

- **Naïve.** Search all deterministic policies

$$\pi : S \to A$$

  - Computationally very difficult
    - The search space has the size $|A|^{|S|}$

Finding the optimal policy

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

- **Policy iteration.** Usually more efficient
  - Initialize $\pi_0(s)$ randomly for all states s
  - Until convergence, do:
    - Evaluate π_i to get the function V^{π_i}
    - Update π_{i+1} to be an improved version

- Recall: Gradient descent, K-means, E-M

State-Action Value Q

- Define the **state-action value** of a policy

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{\pi}(s')$$

- The expected return from:
  - taking an action a at state s
  - then continuing with the policy π

- Satisfies the equality

$$V^{\pi}(s) = \sum_{a \in A} \pi(a \mid s) Q^{\pi}(a,s)$$

Fix

Update

Policy iteration

- Similar to the expectation-maximization, we do:
  - Compute the Q function for the current policy π_i

$$Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^{\pi_i}(s')$$

  - Compute the new policy π_{i+1} for all $s \in S$

$$\pi_{i+1}(s) = \arg\max_{a \in A} Q^{\pi_i}(s, a), \qquad \forall s \in S$$

- **Theorem (w/o proof).** We have $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$
  - Somewhat surprising, as the optimization of π_{i+1} was based on the assumption that we'll use π_i from the next step.

Value iteration

- Policy iteration operates as:
  - computes the infinite horizon value of a policy
    - somewhat heavy – computing value requires iterations
  - improves that policy

- Value iteration is another technique:
  - Compute the best next state, when $H = 1$
  - Compute the best next state, when $H = 2$
  - …

Value iteration

- More formally, the value iteration is:
  - Initialize $V_{(0)}(s) = 0$ for all states s
  - Iterate:
    - Update the value function

$$V_{(i+1)}(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_{(i)}(s') \right)$$

    - Update the policy function

$$\pi_{(i+1)}(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_{(i)}(s') \right)$$

Value iteration vs. Policy iteration

- **Value iteration**
  - Lighter
    - Better for large state-/action spaces
  - Can compute finite-horizon policies

- **Policy iteration**
  - Requires less iteration, usually
  - Guaranteed and stable convergence
    - Especially when γ is very large

Next class

- Q-learning
- Policy Gradient
- Wrap-up

</lecture 24>