Introduction to Reinforcement Learning

CS285 Deep Reinforcement Learning Lecture 2

Kim DongHu

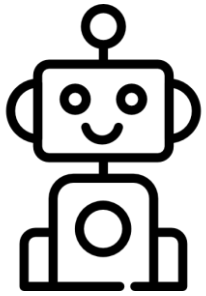
What is RL?

"Maximizing the expected total reward ?
within an environment." ?

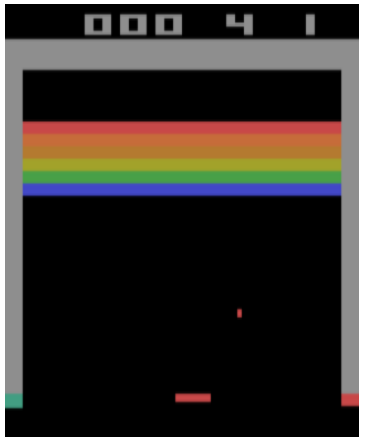
Introductory Examples

Maximizing the expected total reward within an environment.

Player



Game
(Atari Breakout)



Get as much score as possible in Atari Breakout!

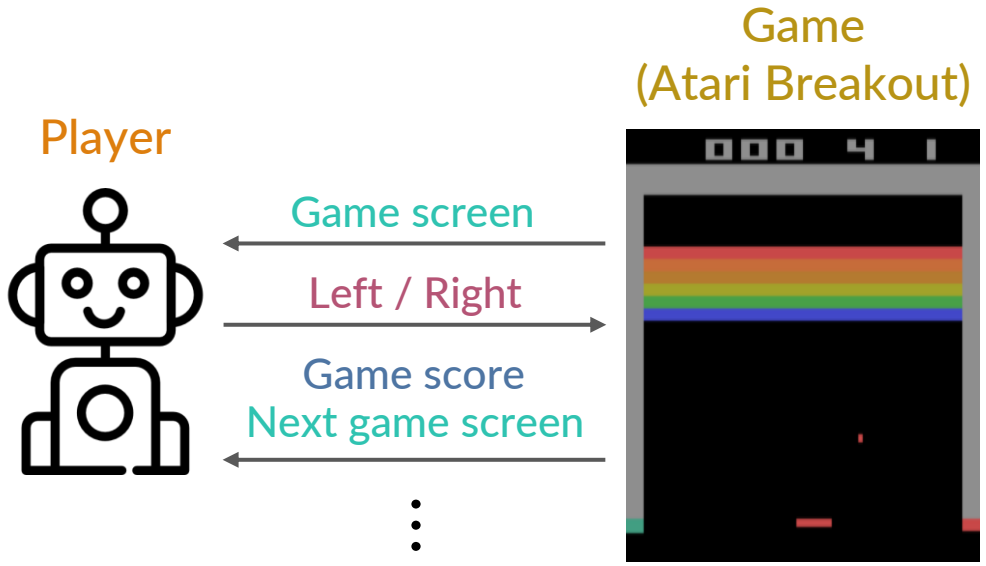
(Robot) Dog

Trainer

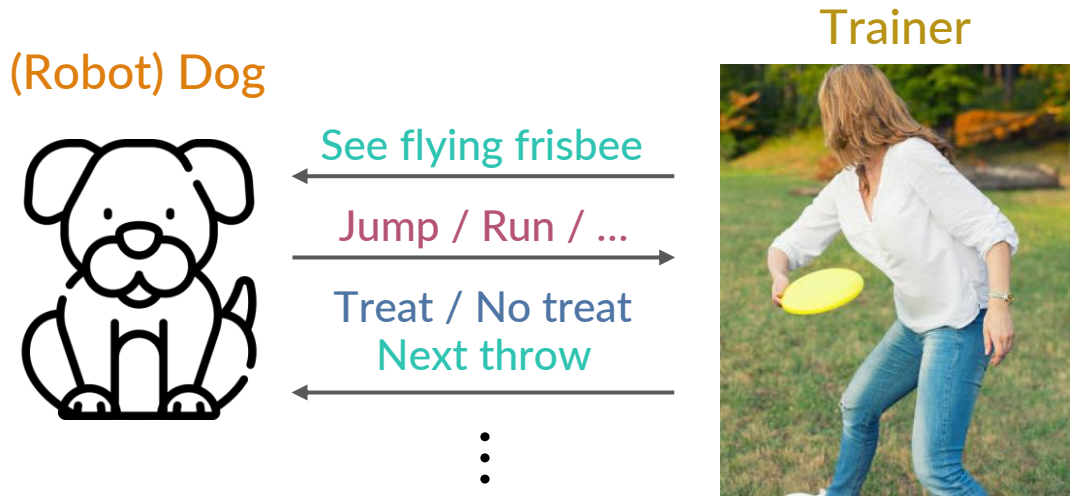
Get as many treats as possible from the trainer!

Introductory Examples

In RL, an agent performs actions given the states/observations, in order to maximize the expected total reward within an environment.



Get as much score as possible in Atari Breakout!



Get as many treats as possible from the trainer!

Markov Chain

- $\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$, where

  $\mathcal{S}$ = Set of possible states

  $\mathcal{T}$ = Transition probabilities

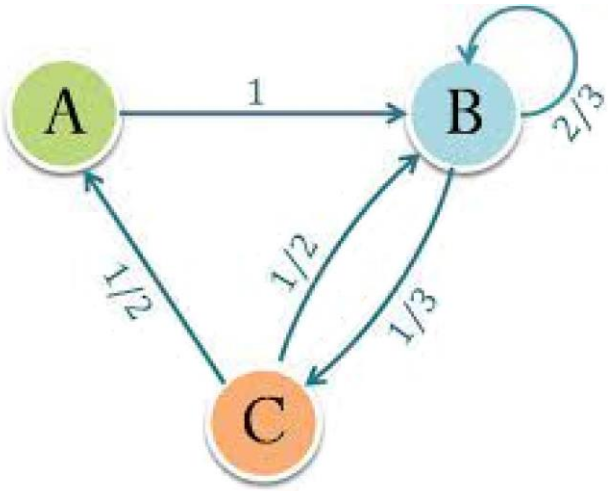
- $\mathcal{T}$ can be expressed as a matrix: $\mathcal{T}_{i,j} = p(s_{t+1} = i | s_t = j)$

  …which can be used as a state transition operator

e.g., $\begin{bmatrix} 0 & 0 & 1/2 \\ 1 & 2/3 & 1/2 \\ 0 & 1/3 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 1/6 \\ 39/54 \\ 1/9 \end{bmatrix}$

state distribution at time $t = 2$

an arbitrary initial state distribution ($t = 1$)



$$\mathcal{S} = \{A, B, C\}$$

$$\mathcal{T} = \begin{bmatrix} 0 & 0 & 1/2 \\ 1 & 2/3 & 1/2 \\ 0 & 1/3 & 0 \end{bmatrix}$$

- But why use this? Markov property!

Markov Property

- A fundamental base of RL: “History doesn’t matter”

  Transitions in $1, \dots, t-1$ do NOT affect future transitions (independence).

Markov property

$$p(s_{t+1} = i | s_t = j, s_{t-1} = j', s_{t-2} = j'', \dots) = p(s_{t+1} = i | s_t = j)$$

- All future theorems and algorithms will be based on this property!
  - Policy Gradient Theorem
  - Bellman Equation
  - ...and many more

Markov Decision Process (MDP)

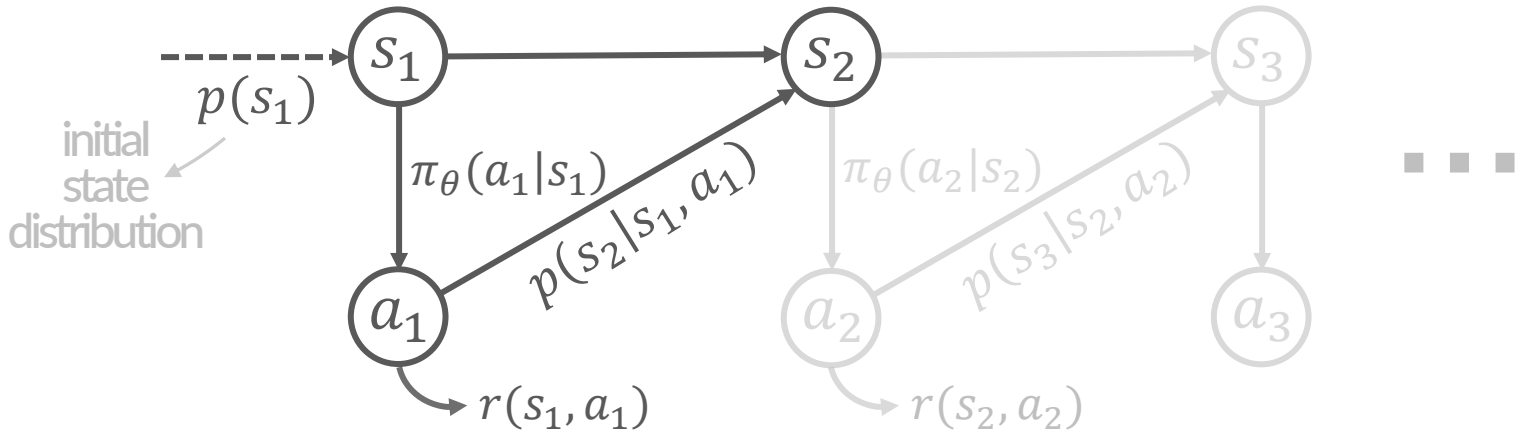
- Markov Chain + Actions + Rewards
- $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$, where $\mathcal{S}$ = Set of possible states $\mathcal{T}$ = Transition probabilities
  $\mathcal{A}$ = Set of possible actions $r(s_t, a_t)$ = Reward function
- An agent has a policy π_θ, which chooses action a given the state s. (θ=Parameters)
  - $a \sim \pi_\theta(\cdot|s)$ for stochastic policy, $a = \pi_\theta(s)$ for deterministic policy.
- Transition is now decided by states & actions: $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$
- The reward $r(s, a)$ tells how good that state-action pair was.
  - e.g., Breakout: Reward +1 for each block break (Dense reward)
    : Reward +100 only on breaking every block (Sparse reward)

Markov Decision Process (MDP)

- Also has Markov property:

$$p(s_{t+1} = i | s_t = j, a_t = k, s_{t-1} = j', a_{t-1} = k', \dots) = p(s_{t+1} = i | s_t = j, a_t = k)$$

State View Example



Temporal View Diagram

Partially Observed Markov Decision Process (POMDP)

- MDP + Observation space
- $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$, where

$\mathcal{S}$ = Set of possible states

$\mathcal{A}$ = Set of possible actions

$\mathcal{O}$ = Set of possible observations

$\mathcal{T}$ = Transition probabilities

$r(s_t, a_t)$ = Reward function

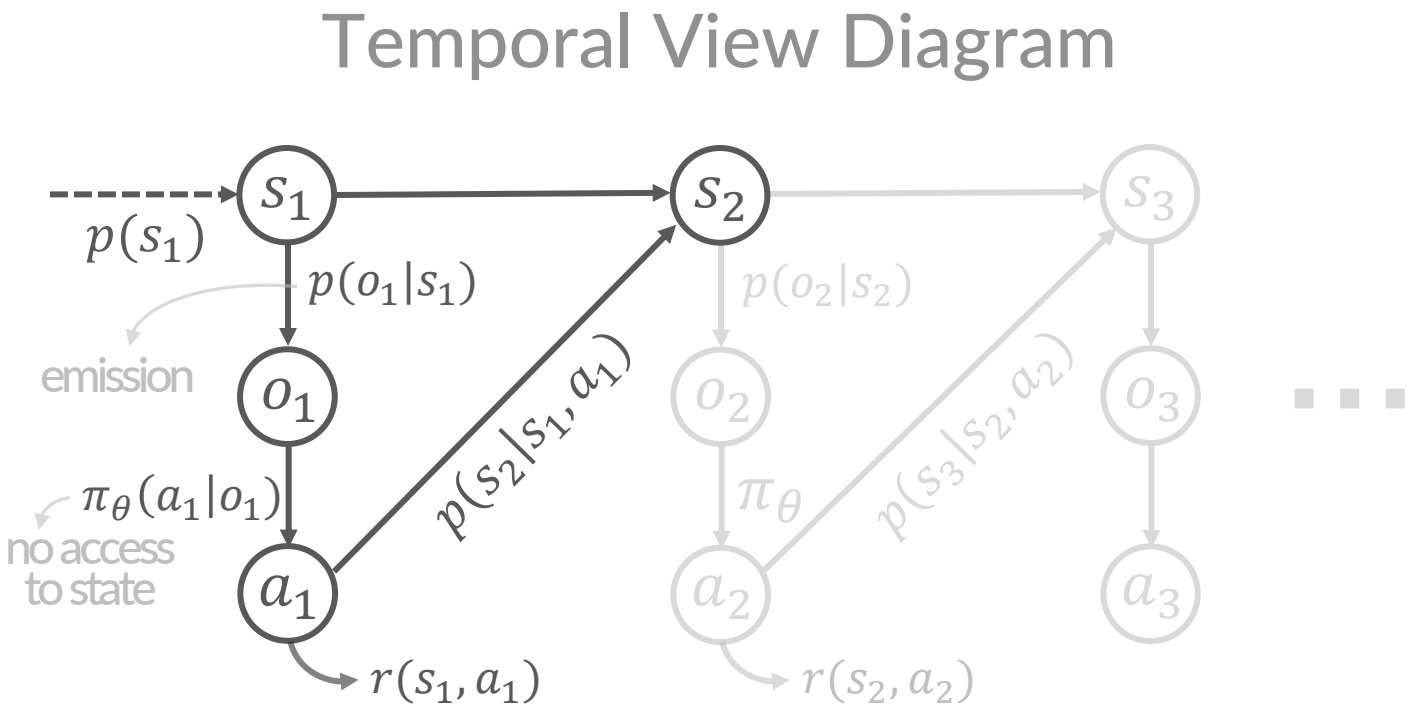
$\mathcal{E}$ = Emission function

- States vs Observations
  - States are 'perfect description' of the environment (e.g., 3D position, motor angle, …)
  - Observations are generated from states via (unknown) emission function (e.g., image observation)
  - Often, the true state cannot be fully induced from the observation.

    (e.g., dog training example – frisbee goes over the fence)

    In fact, both Breakout and Dog Training examples are based on POMDP!

Partially Observed Markov Decision Process (POMDP)



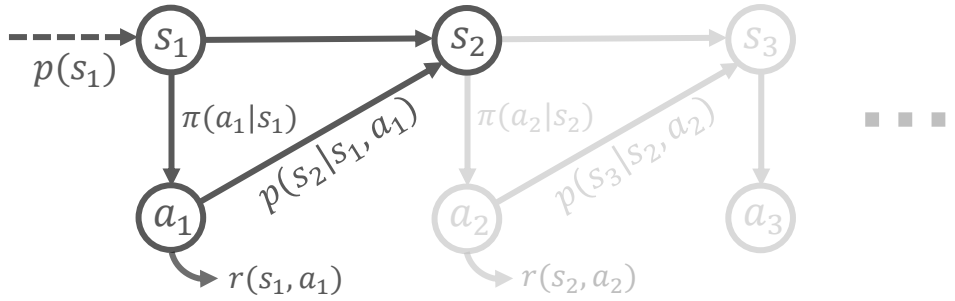
POMDP is commonly used in research, but here we will mostly use MDP.

Again, What is RL?

"Maximizing the expected total reward within an environment."

Sum of $r(s_t, a_t)$?

MDP / POMDP



Expected Total Reward

- Given a (fixed) policy π, there are two ways to compute its expected total reward.

1. Expectation over trajectories
   - Trajectory τ : A single 'episode' of the environment (e.g., one game of Breakout)
   - Each trajectory has its own probability and 'return'. We want to maximize their expected value!

sum of rewards

Objective function of RL (Trajectory based)

$\tau = (s_1, a_1, s_2, a_2, \ldots s_T, a_T)$ where $a_t \sim \pi_\theta(a_t|s_t)$, $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$

Then the probability that π_θ goes through τ is : $p_\theta(\tau) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$.

And the return of τ (total reward sum) is : $r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$.

Our objective, maximize the expected return : $\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)] = \sum_\tau p_\theta(\tau)r(\tau)$.

- This defines the optimal policy: $\theta^* = argmax_\theta\, \mathcal{J}(\theta)$, and $\pi^* = \pi_{\theta^*}$

Expected Total Reward

2. Expectation over state-action pairs
   - A little trickier, but very important as well.
   - Roughly, this dissects trajectories into a bunch of (s_t, a_t)'s, summing their rewards individually.

Objective function of RL (State-action based)

$p(s_t, a_t) \triangleq$ The probability of being in state s at timestep t (in any trajectory), then taking action a.

Then, the expected total reward is now a *sum over all possible timesteps, states, and actions*.

$$\mathcal{J}(\theta) = \sum_{t=1}^{T} \sum_{s_t \in \mathcal{S}} \sum_{a_t \in \mathcal{A}} p_\theta(s_t, a_t) r(s_t, a_t) = \sum_{t=1}^{T} \mathbb{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} [r(s_t, a_t)]$$

   - We see similar bits in both formulations: $\sum_{t=1}^{T}$ and $r(s_t, a_t)$.
     → Both are computing the same sum of rewards, but in a 'different order'!
   - For those curious, $p_\theta(s_t, a_t) = \sum_{s_1 \in \mathcal{S}} p(s_1) \sum_{a_1 \in \mathcal{A}} \pi_\theta(a_1|s_1) \sum_{s_2 \in \mathcal{S}} p(s_2|s_1, a_1) \sum_{a_2 \in \mathcal{A}} \pi_\theta(a_2|s_2) \cdots$
     $\cdots \sum_{s_{t-1} \in \mathcal{S}} p(s_{t-1}|s_{t-2}, a_{t-2}) \sum_{a_{t-1} \in \mathcal{A}} \pi_\theta(a_{t-1}|s_{t-1}) * p(s_t|s_{t-1}, a_{t-1}) \pi_\theta(a_t|s_t)$

Why Expectation?

- A lot of benefits, but smoothness is probably the core.
  - Most reward functions are non-differentiable, let alone continuous (e.g., step function).
  - Taking expectation over these, however, is an easy way to make them differentiable, allowing us to use methods from ML and DL such as gradient descent.
- Example from lecture: Auto-Driving
  - Two actions: Fall (reward -1), Don't fall (reward +1).
  - Assume a single step environment (i.e., $T = 1$).

  - Policy π : $p(Fall|Driving) = \theta,\ p(Don't\ fall|Driving) = 1 - \theta.$ $(\theta \in [0, 1])$
  - Although the reward function is a step function, the expected reward is continuous on θ!
  - Indeed, a gradient 'ascent' on the expected reward will push θ towards 0!

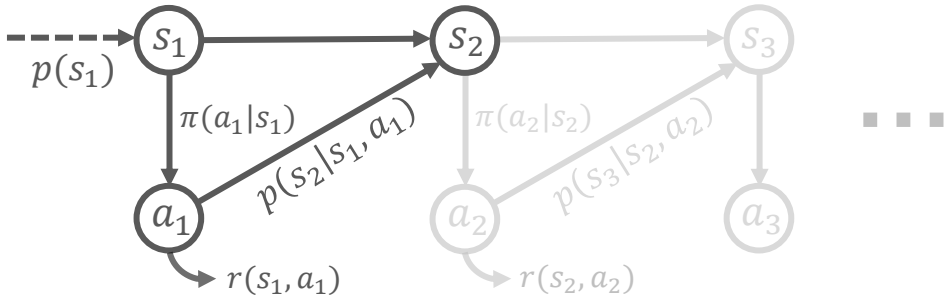
$$\mathbb{E}_\theta[r(s,a)] = (\theta * -1) + \big((1-\theta) * +1\big) = 1 - 2\theta$$

Interlude

- So far…
  - Definition of RL

In RL, an agent performs actions given the states/observations,
in order to maximize the expected total reward within an environment.

  - Definition of the environment (+ Markov property)



  - Definition of objective function in RL

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)] = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)}[r(s_t,a_t)]$$

Interlude

- From now on…
  - Introduction to value functions (an extremely important concept in RL)
  - A rough sketch of algorithms for RL (that will come in future lectures)

- There’s no need to fully understand the next pages.
  - Primary focus on understanding the contents in the first half.
  - The big picture in the next half is absolutely useful, too.

Value Functions

- Value functions evaluate the policy π_θ on specific states and actions.
  - cf) $\mathcal{J}(\theta)$: Evaluates the policy in aggregate.

- Q-function
  - $Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t]$
  - What it computes: "If we take a_t at s_t, then follow π_θ afterwards, what's the expected return?"
  - What it means: "For π_θ, how good was the state-action (s_t, a_t)?" = "Should π_θ prefer (s_t, a_t)?"

- Value function
  - $V^\pi(s_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t] = \sum_{a_t \sim \pi(a_t|s_t)} \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t] = \mathbb{E}_{a_t \sim \pi(a_t|s_t)}[Q^\pi(s_t, a_t)]$
  - What it computes: "If we follow π_θ starting from s_t, what's the expected return?"
  - What it means: "For π_θ, how good was the state s_t?" = "Should π_θ prefer s_t?"

Value Functions

- Value function can express the objective function $\mathcal{J}(\theta)$.

Value function and $\mathcal{J}(\theta)$

$$\mathbb{E}_{s_1 \sim p(s_1)}[V^{\pi}(s_1)] = \mathcal{J}(\theta)$$

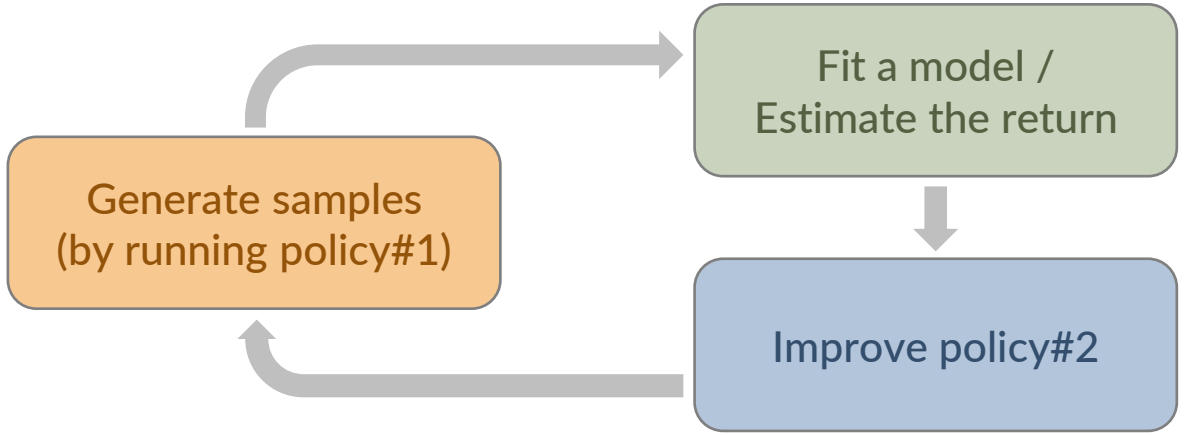
If there's only one initial state (i.e., deterministic) : $V^{\pi}(s_1) = \mathcal{J}(\theta)$

  - What it computes: "If we follow π_θ starting from s_1, what's the expected return?"
    = "From start to finish, how many reward would π_θ get?"
    = "What's the expected return of π_θ?" = $\mathcal{J}(\theta)$

- How do we get this? Where can we use this?
  - Q^{π} and V^{π} can be obtained via Bellman equation.
  - We can use Q^{π} or V^{π} to 'improve' the policy π_θ! → Value-based methods
  - We can use Q^{π} or V^{π} to directly estimate the gradient of $\mathcal{J}(\theta)$! → Actor-Critic methods

Types of RL Algorithms

- A whole catalog of algorithms, each with their own strengths and weaknesses.
  - Policy Gradient: Directly optimize $\mathcal{J}(\theta)$ by gradient ascent, $\theta \coloneqq \theta + \nabla_\theta \mathcal{J}(\theta)$.
  - Value-based algorithms: Focus on estimating value functions Q, V.
  - Actor-Critic: Estimate the value functions Q, V, use it to directly optimize $\mathcal{J}(\theta)$ by gradient ascent.
  - Model-based RL: Focus on estimating the environment itself (transition model).
- No matter the type, they have the same anatomy.



On-policy: policy#1 = policy#2
Off-policy: policy#1 ≠ policy#2

PG: Directly use the returns
Value: Use samples to learn Q, V
AC: Use samples to learn Q, V
Model: Use samples to learn the environment itself

PG: Directly optimize $J(\theta)$
Value: Update policy by 'greed'
AC: Directly optimize $J(\theta)$
Model: Searching(planning), etc.

What Should I Consider?

- Sample efficiency
  - How many samples does the algorithm require?
  - May be a problem when samples are hard to get (e.g., real-life robots)
  - May be a problem even with simulators (e.g., Breakout takes 200M steps = 10+ years of playtime)
  - e.g., Off vs On-policy: Off-policy is more sample efficient but may cause instability.
- Stability / Convergence guarantee
  - Unlike supervised learning, many RL algorithms don't directly optimize the objective function. (The obvious exception being policy gradient methods)
  - With deep neural networks, there's no guarantee that these methods can improve the policy.
  - …although many tricks and hyperparameters can be implemented to make it happen.

What Should I Consider?

- Environment
  - MDP vs POMDP: Use algorithms designed for partial observability, or use recurrence.
  - Continuous vs Discrete action space: Can't use value-based methods on continuous actions!

    e.g., Breakout = Discrete actions (L/R), Robot Dog = Continuous actions (Motor torques)
  - Episodic vs Infinite horizon: The former is often assumed.

- Ease of use
  - Some methods are quite complicated and/or have sensitive hyperparameters.

    e.g., TRPO vs PPO: The latter heavily simplifies the former AND performs better.
  - If unsure, using simpler methods may be a good place to start.

Thank you!