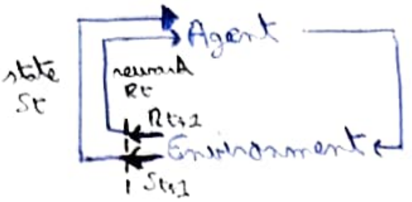
Focus on Reinforcement Learning



input : s_t at each time step
output : R_t at each time step
data : $(s_1, a_1, r_1, \dots, s_T, a_T, r_T)$ ← pick your own actions
goal : learn $\pi_\theta : s_t \to a_t$
to maximize $\Sigma_t r_t$

Reinforcement Learning (RL) : Learn some actions depending on rewards and environment constraints

RL Framework

States / Observations 1 ← more accurate than state because it is often a partial description of the world

Actions & reward function 2

Environment dynamics: 3 Markov decision processes

4 Trajectories and returns

5 Value function & q-function

Policies 6

s_t : The state provide information for action decisions at timestep t (Vector, Matrix, Tensor, ...)

↳ not the same infos given depending on whether s_t is a vector or matrix or tensor or ... (pong game example)

action a_t : at timestep t, the choices the agent can take

↳ can be discrete or continuous (e.g. moving up or down pong game)

Let r be the reward function, then $r_t = f(s_t, a_t)$ a scalar indicating function (pong game ex.)

↳ there is a reward for EACH action not just a group of actions

The environment dynamics of an RL model : Markov Decision Processes (MDP)

- Markov Decision Processes (MDP) : discrete-time stochastic control processes with the Markov property

- Discrete-time : ↳ defines state the agent can be during problem-solving $\xrightarrow{\text{discrete}}$ events are represented as distinct points in time (no "in-between" states).
- Stochastic : Transition distribution : $p(s_{t+1} | s_t, a_t)$ ← the proba to transition to s_{t+1} given the agent is in s_t doing a_t
the agent may not end up in the intended state due to environmental randomness
- Markov Property : The state s_{t+1} depends only on the state s_t (even if the agent has been in many states)
↳ a transition matrix for each action A [illegible] $= p^{A}$

want to apply RL ? → confirm that those 3 criterias have been met

pong game ex. :
- Discrete action (up, down, or more)
- stochastic (the opponent introduces uncertainty in the point of view of the agent)
- Markow property (you move only from your current position to a new one)

A policy maps state to action ← Policy function as a NN for ex. (θ) to optimize are the weights of our NN

↳ Deterministic policy : $a_t = \mu_\theta(s_t)$
↳ stochastic policy : $a_t \sim \pi_\theta(\cdot|s_t)$

} — parameters θ are ajusted to optimize the agent's behavior

A trajectory refers to a sequence of states and actions : $\tau = (s_0, a_0, s_1, a_1, \dots)$

At the end of a trajectory, the agent accumulates rewards. This accumulated reward is called the return $R(\tau)$:

- Finite horizon return ~~[illegible]~~ $R(\tau) = \sum_{t=0}^{T} r_t$ (limited number of timesteps)
- Infinite horizon return $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ (infinite or great number of timesteps)

($\gamma \in [0,1]$ the discount factor ⇒ (↘ γ ⇒ future rewards ↘ valuable)
↖ ensures convergence

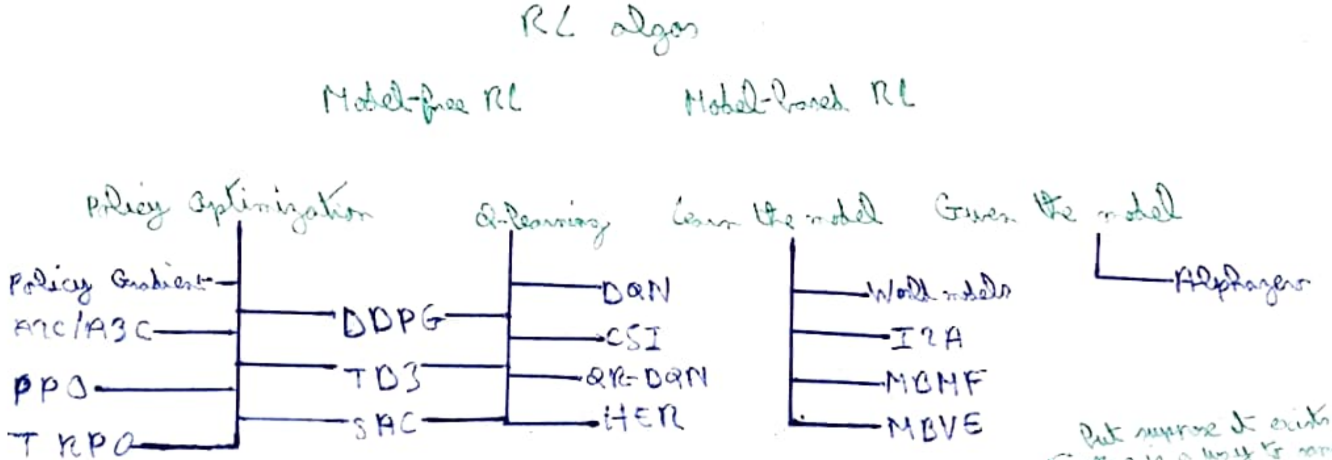
Value function at state s : $V_\pi(s) = E_{\tau \sim \pi}[R(\tau) | s_0 = s]$

↳ the reward we get of the trajectory τ that we expect to follow from policy π, given that $s_0 = s$

Q-function : $Q_\pi(s,a) = E_{\tau \sim \pi}[R(\tau) | s_0 = s, a_0 = a]$

↳ the reward we get of the trajectory τ that we expect to follow from policy π, given that $s_0 = s$ and $a_0 = a$

a Taxonomy of RL Algorithms :



↖ But suppose it exists and there is a way to complete it

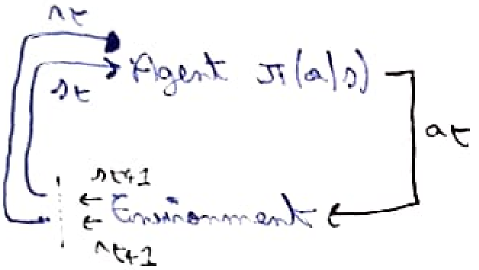
Model-free RL : algos that does not use the transition probability $P(s_{t+1}|s_t, a_t)$ explicitly. They learn policies or value functions based on experience (trial and error)

Model-based RL : algos that attempts to build a model of the environment, including $P(s_{t+1}|s_t, a_t)$, and uses this model to plan and optimize decisions

Gradient Policy algorithm (PG):

- optimize the policy directly (adjust its parameters to improve performance)
- train a policy to act based on observations

Goal: For the policy to choose actions that yield higher rewards



player in s_t → takes a_t based on (π) → the environment/opponent reacts = player in s_{t+1} ↓ takes a_{t+1} based on s_{t+1} ⟵ adjust (π) after a trajectory (τ) using $R(\tau)$

(ex. of this type of agent: basketball player)

(stochastic case)

(ex. : car driving ⟨turning / not stopping ?)

→ often used for continuous action spaces where actions are sampled from a probability distribution rather than being discrete choices

↳ $\pi(a|s)$ = a distribution

{ stochastic policy $\pi(.|s)$ as NN
{ $\pi \sim N(\mu, \sigma)$

⟹ state → NN → (μ, σ)

State — Distribution Parameters

↳ agent sample actions from the constructed distribution

→ then, gradient descent algorithm (ascent here in RL) for in Θ for $J(\pi_\theta)$, the performance measure of a policy $\left(\Theta = \Theta + \underset{\substack{\uparrow \\ LR = \text{[illegible]}}}{\alpha} . \nabla_\Theta J(\pi_\theta)\right)$

RL Optimization problem:
- The RL algo aims to maximize the expected return
- Goal: select the policy that achieves this maximization

- Probability of a trajectory $\tau = (s_0, a_0, \ldots, s_{T+1})$: $P(\tau|\pi_\theta) = P(s_0)\prod_{t=0}^{T}\pi_\theta(a_t|s_t).P(s_{t+1}|s_t,a_t)$

  ↳ $P(A \cap B) =$ ~~[illegible]~~ $P(A) \times P(B|A)$
  $P(A \cap B \cap C) = P(A) \times P(B|A) \times P(C|(A \cap B))$

- Expected return $J(\pi_\theta) = \int P(\tau|\pi_\theta)R(\tau)d\tau = E_{\tau \sim \pi_\theta}[R(\tau)]$ (2)
- Core RL problem: finding $\pi_\theta^* = \text{argmax}_{\pi_\theta}(J(\pi_\theta))$

Why $\nabla_\theta(J(\pi_\theta))$ and not $\nabla_{\pi_\theta}(J(\pi_\theta))$?
1- π_θ is a function: $J(\pi_\theta) = J \circ \pi(\theta)$
2- stochastic policy is itself parameterized by θ: $\pi = \pi(\theta)$ ⟹ (diff θ ⟹ diff π_θ)

thanks to (2), $\nabla_\theta J(\pi_\theta) = \int \nabla_\theta P(\tau|\pi_\theta).R(\tau)d\tau$

↳ identity + log-trick + (2): $\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}[\nabla_\theta \log(P(\tau|\pi_\theta)).R(\tau)]$

(1) + ∇_θ ↳ $= E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log(\pi_\theta(a_t|s_t)).R(\tau)\right]$

Estimation of $\nabla_\theta J(\pi_\theta)$: For $S = \{\tau^i\}_{i \in [1,N]}$ with: N sampled trajectories

τ^i: trajectory obtained from policy π_θ

$$\hat{g} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log(\pi_\theta(a_t^i|s_t^i))\right)R(\tau^i)$$

REINFORCE Algorithm :

1. sample $\{\tau^i\}$ from $\pi_\theta(a_t|s_t)$ (run the policy)
2. $\nabla_\theta J(\pi_\theta) = \sum_i \left(\sum_t \nabla_\theta \log(\pi_\theta(a_t^i|s_t^i)) \underline{R(\tau^i)} \right)$ $\left(\sum_t r(s_t^i, a_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$