Agent and Environment:

step: t (↓ time step) observation = O_t reward = R_t action = A_t

Rewards:

R_t → defines the goal

return (cumulative reward) = $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots$

Values:

state = S

value (the expected cumulative reward, from a state S) =

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$
$$= \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \dots \mid S_t = s]$$

recursively define:

return: $G_t = R_{t+1} + G_{t+1}$

value: $v(s) = \mathbb{E}[R_{t+1} + v(S_{t+1}) \mid S_t = s]$

policy = A mapping from states to actions.

Action values = Also condition the value on actions =

$$q(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$$
$$= \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \dots \mid S_t = s, A_t = a]$$

- Environment (dynamics) • Reward signal (specifies the goal)
- Agent : - Agent state
  - Policy
  - value function estimate ?
  - model ?

Agent State : S_t

history (the full sequence of observations , actions , rewards) :

$H_t = O_0, A_0, R_1, O_1, \ldots, O_{t-1}, A_{t-1}, R_t, O_t$

↓

construct S_t

Fully Observable Environments :

full observability :

observation = environment state

$S_t = O_t$ = environment state

Markov decision processes (MDPs) :

a mathematical framework

A decision process is Markov if $p(r, s \mid S_t, A_t) = p(r, s \mid H_t, A_t)$

Once the state is known, the history may be thrown away

Partially Observable Environments =

Partial observability =

The observations are not Markovian

Using the observation as state would not be Markovian

Partially observable Markov decision process (POMDP)

Agent State =

The agent's actions depend on its state

The agent state is a function of the history =

$S_t = O_t$

$S_{t+1} = u(S_t, A_t, R_{t+1}, O_{t+1})$

where u is a 'state update function'

The agent state is often much smaller than the environment state

State representations =

Deal with partial observability

Examples of agent states =

Last observation = $S_t = O_t$ (might not be enough)

Complete history = $S_t = H_t$ (might be too large)

A generic update = $S_t = u(S_{t-1}, A_{t-1}, R_t, O_t)$ (but how to pick/learn u?)

Policy (defines the agent's behaviour):

A map from agent state to action

Deterministic policy: $A = \pi(S)$

Stochastic policy: $\pi(A|S) = p(A|S)$

Value Function:

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

$$= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s, \pi]$$

discount factor: $\gamma \in [0, 1]$ (tradeoff, immediate vs long-term)

The value depends on a policy

Can be used to < evaluate the desirability of states
select between actions

Value Functions: (recursive forms)

return: $G_t = R_{t+1} + \gamma G_{t+1}$

A Bellman equation:

value: $v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t \sim \pi(s)]$

$$= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(s)]$$

"The expected value of an expected value is just that. It stops being random once you take one expected value, so iteration doesn't change."

Where $a \sim \pi(s)$ means a is chosen by policy π in state s

(even if π is deterministic)

A similar equation holds for the optimal (= highest possible) value:

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

This does not depend on a policy

Model:

A model predicts what the environment will do next

E.g., $\mathcal{P}$ predicts the next state

$$\mathcal{P}(s, a, s') \approx p(S_{t+1} = s' \mid S_t = s, A_t = a)$$

E.g., $\mathcal{R}$ predicts the next (immediate) reward

$$\mathcal{R}(s, a) \approx \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Also stochastic (generative) models

Agent Categories

- Value Based
  - No policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No value Function
- Actor Critic
  - Policy
  - Value Function
- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Optionally policy and/or Value Function
  - Model

Prediction and Control =

Prediction = evaluate the future (for a given policy)

control = optimise the future (find the best policy)

" $\pi_*(s) = \arg\max_\pi v_\pi(s)$ "

Two fundamental problems in RL =

Learning = < The environment is initially unknown
The agent interacts with the environment

Planning = < A model of the environment is given (or learnt)
The agent plans in this model (without external interaction)

Learning Agent Components

All components are functions

Policies: $\pi = S \to A$ (or to probabilities over A)

Value functions: $v = S \to \mathbb{R}$

Models: $m = S \to S$ and/or $r = S \to \mathbb{R}$

State update: $u = S \times O \to S$

Can use neural networks, and use deep learning to learn

Take care = we do often violate assumptions from supervised learning (iid, stationarity)

$I(\cdot)$ = the indicator function = $I(\text{true}) = 1$ and $I(\text{false}) = 0$

Agents can learn a policy, value function and/or a model