(6) The RL framework : The problem

↓

RL basically involves an agent learning to interact with its environment to maximise some kind of reward signal

↓

agent is the one learning from trial & error, on how to behave in the env. to maximise reward.

↓

Assumptions : a) time evolves in discrete timesteps

b) At T_0 , agent observes the env.

↓

and the agent is able to fully observe what ever state the environment is in at present.

↓

i) like the configuration of a chess board.

ii) like the state of a go game.

↓

So, the agent receives a state of the env. at time step T_0, after observing which, it takes some action, based on which it gets a corresponding reward, and an updated state.

↓

This process goes on and it is represented as a sequence of states, actions and rewards as follows :

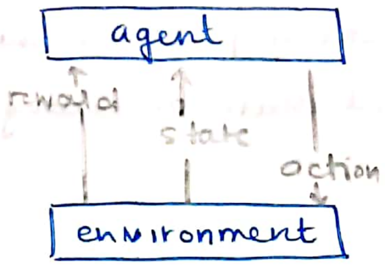
↓

$\{ S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots\ldots, S_n, A_n, R_{n+1} \}$

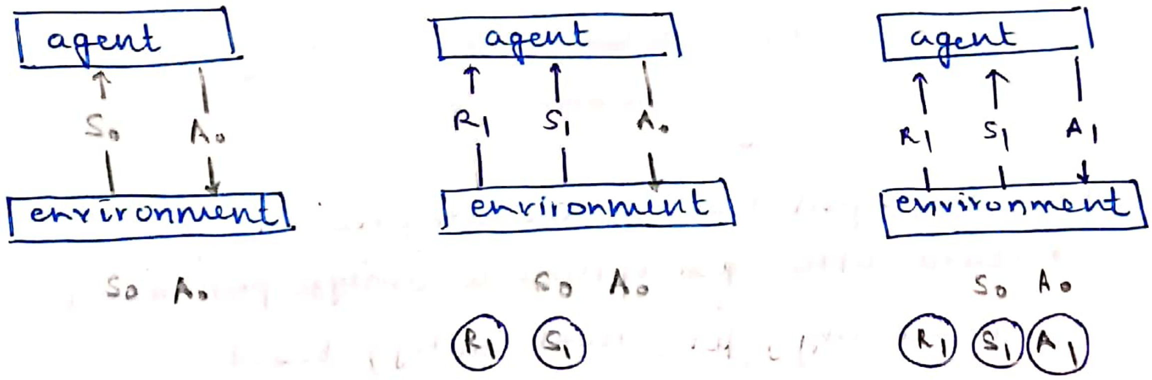
As stated earlier, the goal of any agent is to maximise the reward (cumulative expected reward). Hence the sequence of actions (or strategy) should be chosen accordingly to maximise the reward

↓

The knowledge to be able to choose an action, at a given situation (state) comes from the interaction with the environment.



→ Basic interaction b/w the agent & the environment.



↓

A mathematical model for a real world RL problem can be defined using the following:

i) states (possible states)
ii) actions. (possible actions)
iii) rewards. (possible rewards)
iv) rules of the environment
(probabilities of actions being taken given a certain state, and the probability of a resulting state taken given a certain action)

⑥ An instance of a RL problem can be called a task.

↓

if playing a game is the problem, one chance can be called a task for the problem.

↓

There are two types of RL problems based on the type of tasks involved in the problem:

1) episodic : tasks of problems which have a defined terminal state or ending time step.

for example a chance of the game ending

SD car reaching its destination

robot crashing while learning to walk

a game of chess

↓

Here the agent looks at the total reward received after each episode to analyze performance and accordingly can tune their strategy, based on which it can perform in the next episode, and so on.

↓

over many episodes the agent starts to make better decisions for itself, in order to choose a strategy to increase cumulative reward (which can be the game score in case of a game)

↓

sequence of states will have an end.

↓

$\{ S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots S_T \}$

Scanned by CamScanner

ii) continuous tasks : donot have an ending timesteps

↓

i.e interaction continues with the environment without a pre-specified limit.

↓

for example an algorithm learning to buy and sell stocks in response to the financial market

↓

Here the agent needs to optimize the strategy on the go.

↓

* Reward Hypothesis :

↓

In RL, for any problem, a reward signal is given, (which could be a function or formula), maximization of which results in the completion of some specified goal

↓

this is the reward hypothesis

↓

in case of the puppy, it was quite straight forward to figure that maximization of the number of treats works, but that won't always be the case. for example, a robot learning to walk.

↓

judging its performance only by viewing can be very subjective, depending on the viewer, but if we have some kind of mathematical function, which can act as a parameter to judge the quality of walk, then that would be quite uniform.

↓

this function is called the reward function

Maximization of this reward function is what will also help us realize the goal of learning to walk for the robot.

↓

Analyzing DeepMind's work on this

↓

goal : robot to stay walking as long and quickly as possible while exerting min. effort.

↓

To specify or frame this problem, we require:

i) states
ii) actions
iii) rewards.

↓

i) states : context provided to the robot (agent) to take intelligent decisions which can include:

a) current position & velocity of joints.

b) information about the surface

↓

flat?, inclined?, large step?

c) ~~contact~~ contact sensor data

↓

to check whether robot is standing or has crashed

(one of the many uses)

ii) actions : Decisions to be made in order for the robot to walk. As movement is controlled by joints and the pressure exerted on them, ∴ the action can be considered to decide the amount of pressure to put on joints, since that controls both velocity and position of the joints.

iii) reward signal / function :

* This is an episodic task i.e. it continue till the robot doesn't fall

$$r = \min(v_x, v_{max}) - 0.005(v_y^2 + v_z^2) - 0.05y^2 - 0.02\|u\|^2 + 0.02$$

↓

Here r gives the reward for one time step, the cummulative reward would be the summation of reward received at all time steps.

↓

$0.02 \rightarrow$ reward for not falling down

↓

$\min(v_x, v_{max}) \rightarrow$ reward for being able to walk forward. The faster the robot walks, the more reward it receives, but upto a certain limit. (adjusted using v_{max})

↓

$-0.005(v_y^2 + v_z^2) \rightarrow$ penalizes movement in other axes. {any velocity induced} → in those axes

↓

$-0.05y^2 \rightarrow$ penalizes displacement in axis ⊥ to the one required.

$-0.02\|u\|^2$ → penalty for higher force applied

↓

higher force applied results in big difference in the velocities & hence the position of the joints

↓

viewed by the erratic movement of the joints

↓

highly undesirable

↓

but if the force applied is less, the movement will seem more stable.

Now, as per the reward hypothesis, maximization of the reward signal will result in the robot learning to walk.

* ~~C~~ Cumulative reward → Till now, we know the reward function gives the reward for a single time step.

↓

from this to maximized cumulative reward

↓

maximize over all time steps ?

i.e. going for maximum reward for each time step.

not the best strategy, since in order to maximize the reward at one time step, it can happen that the robot gets so destabilized that it crashes.

↓

so the robot should be able to take future time steps into consideration while deciding apt action for current time step, in order to maximize cumulative reward.

↓

can be thought of how actions in real life have short term & long term consequences, so it is important to keep in mind both, while deciding the action for current time step.

↓

incase of robot: Agent has long term stability in mind, hence might decide to walk slowly in order to avoid falling down in subsequent time steps, sacrificing short term reward to improve long term reward.

↓

why? ✓

how? → summation of the reward and the reward from future time steps

↓

expected reward for any arbitrary time step. ← rewards can't be known with certainity for future time steps

↓

$G_t = R_{t+1} + R_{t+2} + \dots$

↓

it is logical to assign weights to the future time steps

↓

the ones coming sooner should have higher weight. → $G_t = R_{t+1} + \gamma R_{t+2} + (\gamma)^2 R_{t+3} + \dots$

14

In the last equation, $G_t \rightarrow$ discounted return

should be nearer to 1 than 0, to avoid short sightedness. $\leftarrow$ γ $\rightarrow$ discount rate (more relevant to continuous tasks)

$\downarrow$

$\gamma \rightarrow 1$ (all rewards have equal weights)

$\downarrow$

$\gamma \rightarrow 0$ (immediate reward matters)

* Markov Decision Process

↓

used to define any RL problem rigorously

↓

recycling robot example → robot designed for picking up empty soda cans, equipped with arms to grab the cans, and runs on rechargeable battery.

↓

Here the robot needs to decide, when to go searching for cans to collect, and when to get its battery charged at the docking station.

↓

Action space or A ← Actions : i) stay put (some one else brings the can to it, acting like a dust bin)

ii) search rooms for cans

iii) getting battery charged.

without terminal states

State space or S ← States : level of the battery of the robot

i) low

ii) high

↓

S^+ → with terminal states.

↓

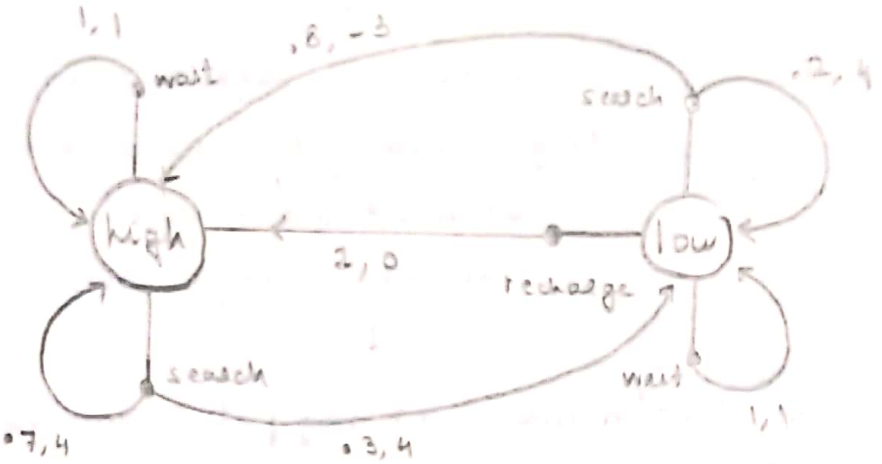
Now, if the state is high, it makes sense for the robot to stay put or search for cans, but not much to getting its battery recharged.

↓

while when it is low, searching might be risky, since it might get discharged, being left stranded and would require human intervention which is not encouraged.

from the above reasoning, it can be said that not always would an action be applicable for certain state, hence action space can be defined per state

state
↓
$A(s)$
↓
Action space for states



(markov property)
one step dynamics : given a state, probabilities of reaching the (other) next state on a certain action.

↓

$$p((s', r)/(s,a)) = P\left(\frac{S_{t+1} = s', R_{t+1} = r}{S_t = s, A_t = a}\right)$$

↓

$$P\left(\frac{\text{high}, 4}{\text{high, search}}\right) = 0.7$$

↓

probability that the state at next time step is high and reward is 4, given current state is high and search action is performed.

↓

These only take the current time step into consideration, & not the evolution steps till now.

one step dynamics are used to better explain the problem by associating certain probabilities with actions to see the resulting state and the corresponding rewards.

↓

important to be able to define the problem

↓

the agent does not know about it.

* So any RL problem can be defined a finite MDP. i.e.

(both state and action space, finite)

a) state space

b) action space

c) rewards

d) one step dynamics.

e) discount rate: