Unit 4 - Semi Supervised learning & Reinforcement learning

1. Markov Decision Process (MDP)
2. Bellman equations
3. Policy evaluation using Monte Carlo
4. Policy iteration and Value iteration
5. Q-learning
6. State-Action-Reward-State-Action (SARSA)
7. Model-based Reinforcement learning.

Markov Decision Process (MDP) :→

→ It is a feedback-based Machine learning Tehnique in which an agent learns to behave in an environment by performing the actions and seeing the result of actions.

- for good action - positive feedback
  bad action - negative feedback.
- Agents learn automatically using feedback without any labeled data.

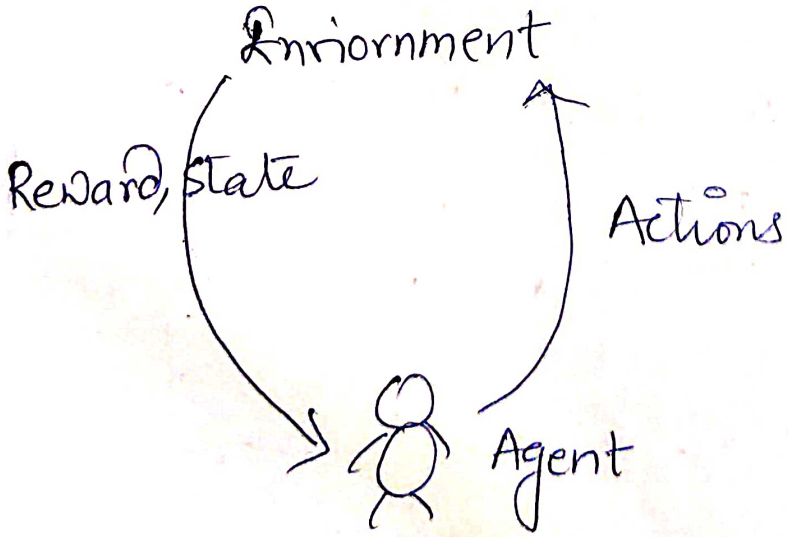
- Agent learn by their experience.
- RL used for sequential decision making and for long term goals.

like - Game playing, Robotics.

Definition:- Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that.

eg:- Robotic dog - arm movement learn

- It's a part of AI & AI agent
- Agent do these three things
  1. take action
  2. change state/remain in the same state
  3. Get feedback.



Terms used in Reinforcement Learning :→

① Agent : entity preceive/explore the enviornment and act upon it.

② Enviornment: Situation in which agent is present or surrounded.

③ Action : moves by agent in enviorment.

④ State : situation returned by enviorment

⑤ Reward : feedback returned.

⑥ Policy : strategy applied by agent for next action.

⑦ Value : long-term retuned

⑧ Q-value : Similar to value, take one additional parameter

Key features of Reinforcement Learning :—

① Instruction not given to agent about enviornment and what action need to be taken.

② Based on hit & trial proces.

③ Agent takes the next action and changes states according to the feedback of the previous action

④ Agent get a delayed reward.

⑤ Enviorment is stochastic, agend explore to get max positive reward

Approaches to implement Reinforcement Learni

Three way to implement reinforcement in ML

① **Value based:-**

- It find the optimal value (maximum) at a state under any policy.

② **Policy-based:->**

It find optimal policy for the maximum future rewards without using the value fun.

It use 2 policy

(i) **Deterministic:->**

Same action at all state

(ii) **Stochastic:->**

Probability determines the produced action.

③ **Model-based:->**

Virtual-model is created and agent learn from the enviorment by exploring it.

Elements of Reinforcement learning :→

4 main elements of RL.

1. Policy
2. Reward signal
3. Value fun
4. Model of Environment

→ Policy means how agent is behaving at a given time. It map previous states of environment to the action taken on those states.

→ Core element of RL.

Deterministic policy $a = \pi(s)$

Stochastic policy : $\pi(a|s)$

$= P[At=a \mid st=s]$

② Reward Signal :-

- RL use reward signal on each action.
- If good action & bad action is there, award signal are send to agent accordingly.

- agent goal is to maximize the total no. of rewards for good actions.
- If rewards are bad, agent change the policy.

③ Value fun^c:→

Value fun^c gives info about how good the situation and action are.

- A reward indicates the immediate signal for each good and bad action, where value fun^c specifies the good state and action for the future
- Value fun^c depend on Reward
  if no reward — no value

④ Model:→

- It tell the behavior of the enviornment.
- model is used for planning,
- Mode make inference about the enviornment will behave.
- If state and action are given model can predict the next state and reward.

How RL work?

first understand with the help of example.

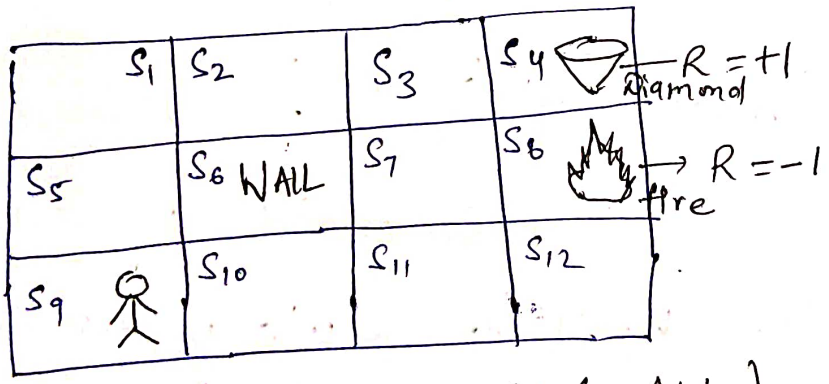
RL need two main things

① Enviornment:-

eg: Room, maze, football ground.

② Agent

like AI robot.



→ Agent can't cross S6 (WALL)

if reach S4 → Reward +1

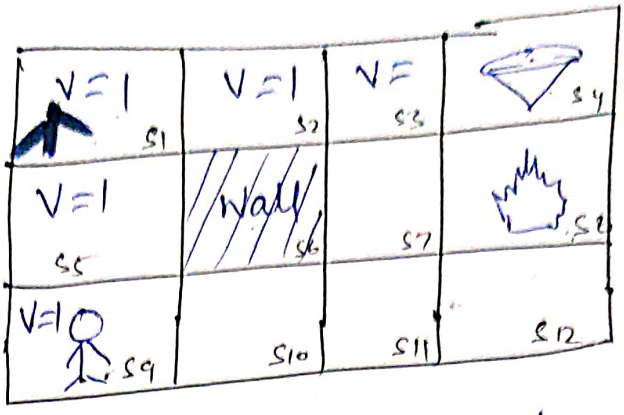
Reach S8 → Reward -1

→ only four action → move — up, down, left, right

→ He have to use minum/fewer steps.

Let path S9 - S5 - S1 - S2 - S3 +1 Reward.

So when he will go he will put value 1 on the previous steps.



- But what will happen when he start moving from the block, which has 1 value at both side (S1)
- from S1 → he can moov S2 and S5

- So this approach is not good for solution
- Bellman equation, concept used behind the reinforcement learning.

The Bellman Equation :-

- This algo is associate with dynamic programming and useed to caliulate the value of a decision problem at a ~~term~~ certain point by including the values of previous states.

- Key elements used in Bellman eq:
  1. Action performed by agent 'a'
  2. state occured by performing action is 's'

DETAILED LECTURE NOTES (5)

- Reward / Feedback 'R'
- discount factor is Gamma 'γ'

Bellman eq:-

$$V(s) = \max [R(s,a) + \gamma V(s')]$$

- $V(s)$: Value Calculate at a particular point
- $R(s,a)$: Reward at a particular state s by performing an action (Instantaneous reward)
- γ : Discount factor (0-1)
- $V(s')$: The value at the previous state (futuristic Reward)

Let 1st block:

$$V(s3) = \max [R(s,a) + \gamma V(s')]$$

$V(s') = 0$

$V(s3) = 1$

Let $\gamma = 0.9$

2nd Block

$V(s2) = 0 + .9 = .9$ (no Reward)

3rd Block

$V(s1) = (0 + 0.9 \times 0.9)$

$= .81$

4th Block

$V(S5) = .9 \times .9 \times .9 = .73$

$V(S9) = 0.66$

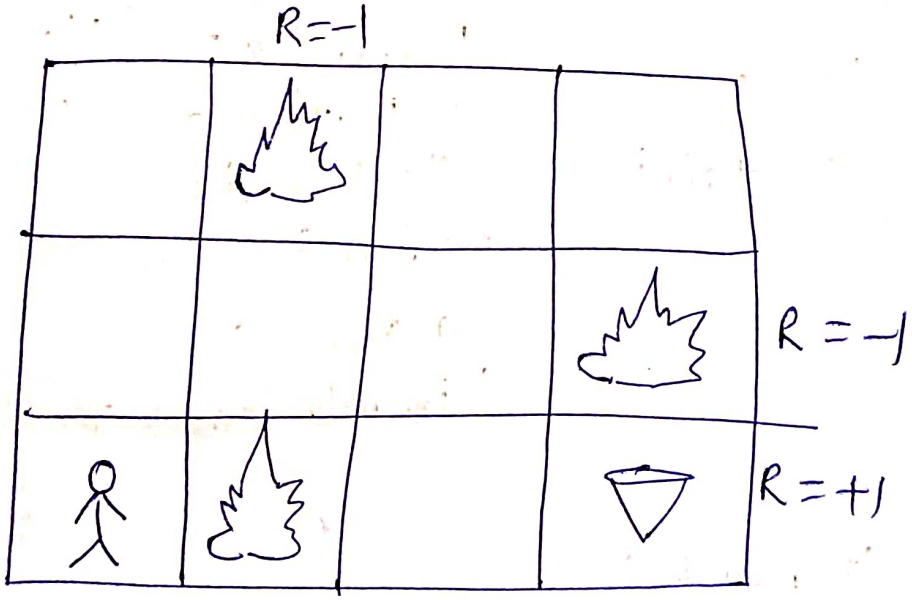
Calculate for rest value

$S7 = 0.9$

$S10 = 0.73$

$S11 = 0.81$

$S12 = 0.73$



Types of Reinforcement Learning:→

2 type of RL

① Positive R
② Negative R

→ In this strength and freq. of the behavior is increased. (Add something to increase the tendency that expected behaviour would occur again)

Advantage of Positive RL:-

① Maximize Performance
② Sustain change for a long period of time
③ Too much reinforcement can lead to an overload of states which can diminish the results.

→ It is opposite of P-ve RL.
It increase the tendency that the specific behavior will occur again by avoiding the negative condition.

Advantages of R-ve RL:-

① Increase Behavior
② Provide defiance to a min stadard of performance
③ It only provides enough to meet up the min behavior

How to represent the agent state?

- We can represent the agent using Markov state that contain all the required info from the history
- State S_t is Markov state if it follows the given condition:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1 \dots S_t]$$

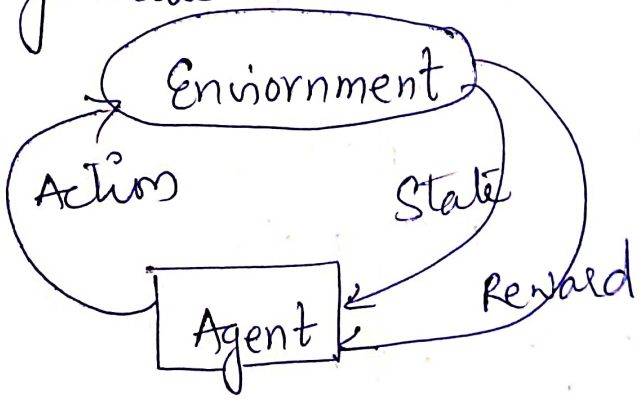
- Markov state follow Markov property which state that "future is independent of the past and can only be defined with the present".

- RL work on fully observed enviornments, where the agent can observe the enviornment and act for new state ⟶ this whole process is known as Markov Decision process.

Markov Decision Process: →

MDP is used to formalize the reinforcement learning problems. If the environment is completely observable, then its dynamic can be modeled as a Markov Proces

- In MDP, agent constantly interacts with the environme and perform actions; at each action, the enviorment respond and generates a new state.



MDP contain a tuple of four elements (S, A, P_a, R_a

(1) State (S)

It is a set of tokens that represent every state that the agent can be in

(2) Action (A)

- Set of all possible Action,
- A(S) defines the set of action that can be taken being in state S.

③ Reward (Ra)

Ra indicates the reward for simply being in the state S.

R(S, a) — Reward for being in a state S and taking an action a.

R(S, a, S') → Indicates the reward for being in state S, taking an action a and ending in a state S'.

④ Policy:→

A policy is a solution to the MDP. A policy is a mapping from S to a,

eg:- States : S

Model : T(S, a, S') ~ P(S' | S, a) → probability to reach S' state by a action from S state

Actions : A(S), A

Reward : R(S), R(S, a), R(S, a, S')

Policy : $\pi(S) \rightarrow a$

π^*

MDP uses Markov property

Markov Property:-

If the agent is present in the current state S1, performs an action a1 and move to the state S2, then the state transition from S1 → S2 only depends

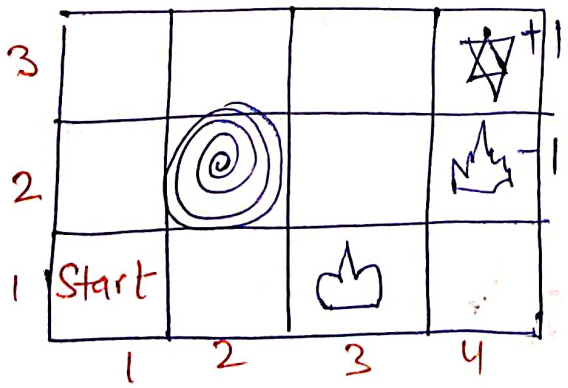
on the current state and future action and states do not depend on past actions, rewards or states.

eg:- Chess game — player only focus on the current state and do not need to remember past actions or states.

Markov Process:-

MP is a memoryless process with a seq. of random states $S_1, S_2 \ldots S_t$ that uses the Markov property. Markov process is also known as Markov chain, which is a tuple (S, P) on state S and fun P.

eg:-



Two solun exist: ① R R U U R from (start)

② U U R R R

Prob of UP = 0.8
Left = 0.1
Right = 0.1

0.8+0.8+0.1+0.1+0.1 =

Policy Evaluation using Monte Carlo:—

Any method which solves a problem by generating suitable random numbers, and observing that fraction of no. obeying some property or properties can be classified as Monte-Carlo method.

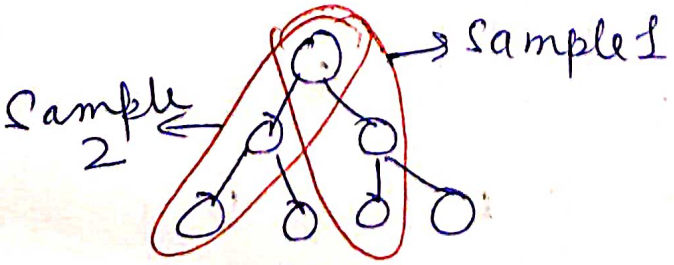
- Monte carlo method are ways of solving the RL problem based on averaging sample returns.
- We have policy π and accordingly calculate
  state_value_func (V_π(s))
  action-value-func (q_π(s))

So state value is cumulative sum of rewards from the starting state to end state by following a policy π.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots \gamma^{T-1} R_T$$

eg: video Game (seq. of all states, frames from start to end game).

- In Monte-Carlo we explore all states from the starting state



→ then avg all the returns.

Two approaches
① first visit
② Every visit

- Keep counter N(s) for each state s
- S(s) – sum of diff return (cumulative)

First visit MC policy evaluation

Initialize:

$\pi \leftarrow$ policy
$V \rightarrow$ state value
Returns (s) $\leftarrow$ empty list $s \in S$ $S_s = 0$ initialy. $N_s = 0$

Repeat forever for (k episodes)

Generate episode using π

for each state s appearing in episode

When reach at that state → $N_s = N_s + 1$ $G \leftarrow$ return first occurence of s

$S_s = S_s + G_t$ Append G to Returns (s)

$V_s = \frac{S_s}{N_s}$ V(s) $\leftarrow$ avg (Return (s))

Problems of Monte carlo: →

① Work for episods only not for continuous or infinite problems
② Need to complete an episode first before we can update all values.

Policy iteration and Value iteration →

- Bellman expectation eq. for state value under policy π

$$V_\pi(s) = \sum_a \pi(a|s) \sum p(s', r | s, a) [r + \gamma V_\pi(s')]$$

- $V_\pi(s)$: Value of a state
- $\pi(a|s)$: probability of taking action a in state s
- $\sum p(s', r | s, a)$: probability for next state
- Recursive relation b/w current state's value & successive state value

In this equation adding all values so eq. is updated

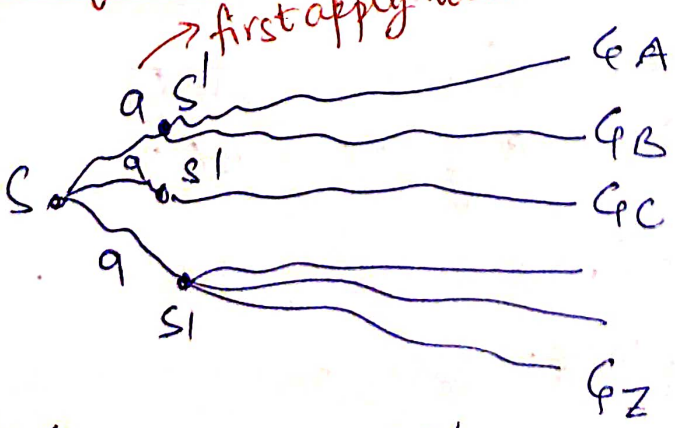
$$V_{k+1}(s) = \sum_a \pi(a|s) \sum p(s', r | s, a) [r + \gamma V_k(s')]$$

eg: Value of state

Total 26 discount episods

$$\therefore V_\pi(s) = \frac{1}{26} \sum (G_A + G_B + \dots + G_Z)$$

② Value of action:-



$$q_\pi(s, a) = E_\pi[G_t | S_t = S, A_t = S]$$

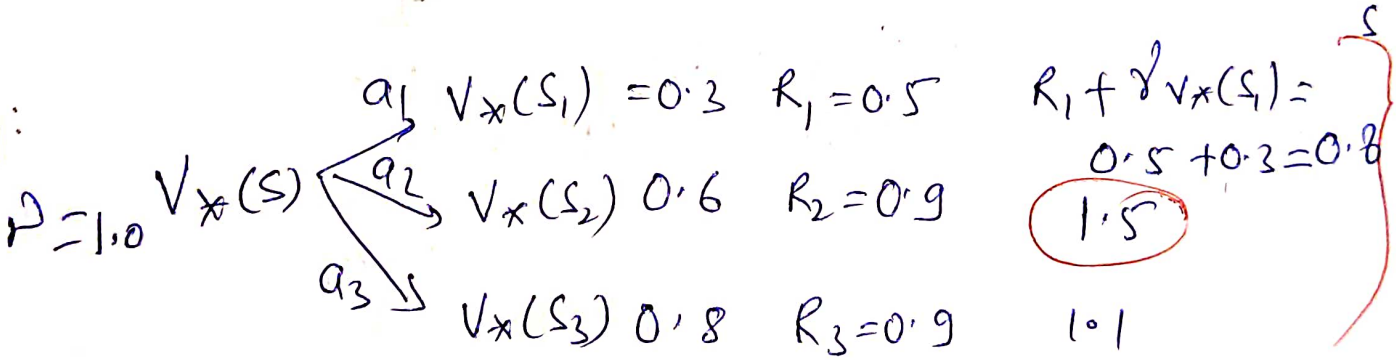
$$\hat{q}_\pi(s, a) = \frac{1}{26} \sum (G_A + G_B + \dots G_Z)$$

- What is optimal state value and optimal action value.

$$V_*(s) = \max_{\pi} V_\pi(s)$$

$$q_*(s,a) = \max_{\pi} q(s,a)$$

eg:



$V_*(s) = 1.5$

→ How to find optimal policy

Policy → what action to take in a certain state?

Ans: ① Policy iteration

② Value iteration

Policy iteration

- It has 2 parts

① Policy evaluation – compute state value for all states
$V_\pi(s)$ ↑ π_{new}

② Policy improvement – Improve the policy

for this

① Set random policy π_0
↓ evaluate it

② find state value using π → V_{π_0}
↓ improve

③ find new policy and repeat until good policy. (optimal) → π_1
↓ & evaluate
V_{π_1}
↓ improve
⋮
π_*

Steps:-

① Set π as evaluate policy

② Set arbitrary value for all states

③ Using π compute new values for the states

$$V_{new}(s) \leftarrow \sum_{s', R} P(s', R \mid s, \pi(s)) [r + V_{old}(s')]$$

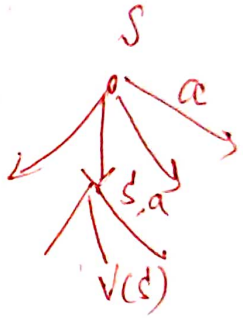
for all states

④ Repeat from 3 until convergence of the state values.

- After that calculate the policy improvement.

$$\pi(s) \leftarrow \arg\max_{a} \sum_{s', r} P(s', r \mid s, a)[r + \gamma V(s')]$$

Value iteration :-



1. Compute value of state using action until all value convergence
2. find optimal value after that
3. Return new policy

- In this we are not calculating policy again and again only after optimal value, calculating single time only.

Limitations of 2 algorithm :-

1. Synchronous update
2. Need knowledge of the enviorment dynamics
3. Not learning from experience

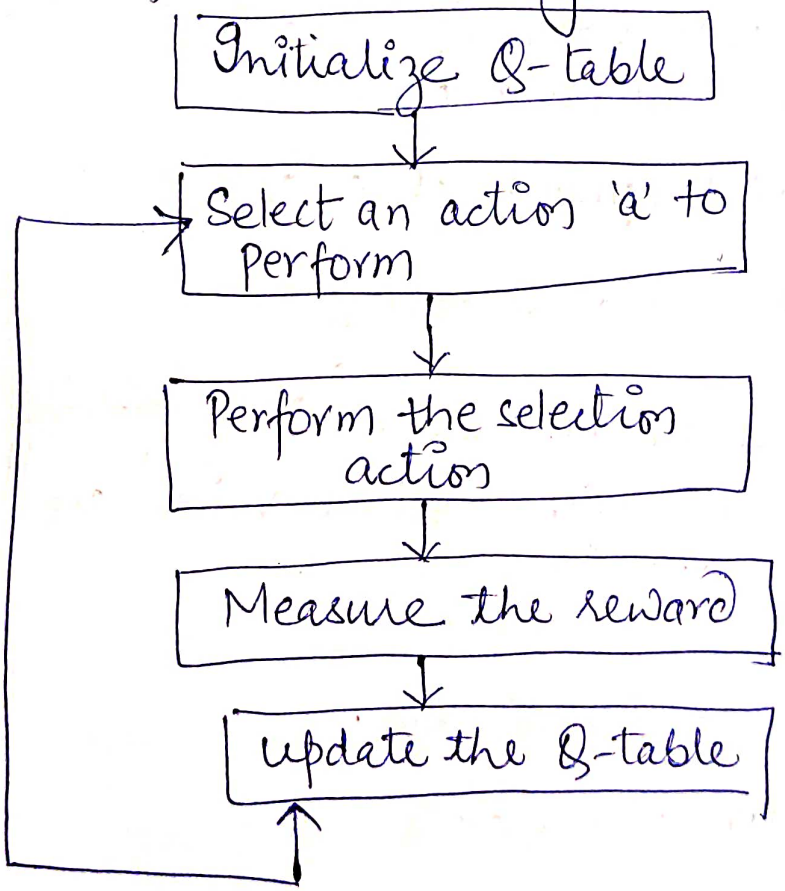
Q: Difference b/w Policy iteration and Value iteration

Reinforcement Learning Algorithms:-

- RL algo are basically used in AI applicat and gaming application.
- Main algorithms are
  1. Q-Learning
  2. SARSA (State Action Reward State Action)

Q-Learning:->

- It is a off policy RL algorithm, used for the temporal difference learning.



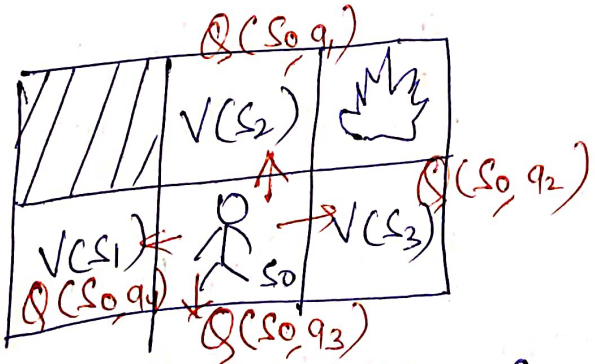
- The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumtances.

4/ applica

- find the best action to take at a current state.
- Goal of agent in Q-L is to max the value of Q.
- Value of Q-learning can be derived from the Bellman eq.

$$V(s) = \max\left[R(s,a) + \gamma \sum_{s'} P(s,a,s')\, V(s')\right]$$

(end state; Reward; Discount factor; Probability)



- This MDP so agent only cares for the current state and future state.

→ Q represent quality of action at each state.

- So instead of writing value, we are using pair of state & action $Q(s_i, a_i)$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')\, V(s')$$

$$V(s) = \max[Q(s,a)]$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max Q(s',a'))$$

- A Q table or matrix is created while performing the Q-learning. It contain state & action pair and initializes the values to zero.

② SARSA (State Action Reward state Action)

- It's a slight variation of the popular Q-learning algo.
- In RL, It use 2 policy
  - ① On policy:- Agent learn the value func according to the current policy
  - ② Off policy:- Agent learn the value func according to the action derived from another policy.

Q-learning is off-policy technique and uses the greedy approach to learn Q-value.

- SARSA used on-policy technique and uses the current policy to learn the Q-value.

eg:

$$\text{QLearning} = Q(S_t, a_t) = Q(S_t, a_t) + \alpha\left(R_{t+1} + \gamma \max Q(S'_{t+1}, a') - Q(S_t, a_t)\right)$$

$$\text{SARSA} = Q(S_t, a_t) = Q(S_t, a_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t)\right)$$

(current state → S_t, current action → a_t, Reward → R_{t+1}, next state → S_{t+1}, next action → a_{t+1})

(SARSA) (s, a, r, s', a')

PAGE NO. 7

Model-based Reinforcement Learning:→

- Let your are playing chess.
- U made a move, then ur opponent makes a move.
- After few move u realize that your 1st move was not correct.
- So, next time we think about the opponent next move and creat a space-tree in our mind and made a best move.

- Now if we replace ourself by AI-agent then you get Mode-based Reinforcement learning.

Model

MDP (S, A, P, R)

In model M_i [S, A, P_i, R_i]

- So model M_i going from state s to s' after performing action A, is subject to the probability $P_i(s'|s,A)$ have Reward $R'(r'|s,A)$.

Model is anything the agent can use to predict how the enviornment will respond to its actions, concretely the state transition $T(s'|s,a)$ or reward fun $r(s,a)$.

We have 2 type of models.

Model-free RL

- We perform sampling and simulation to estimate rewards.
- They perform actions either in the real world or in compt and reward $\leq$ +ve / -ve

Model-based RL

- If we define a cost func ourselves, we can calculate the optimal actions using the model directly.
- It reduce no. of interaction with the real enviornment during the learning phase.

- Main loop of Model-based RL