Lecture 12: Deep Reinforcement Learning (Part1)

Reinforcement Learning + Deep Learning

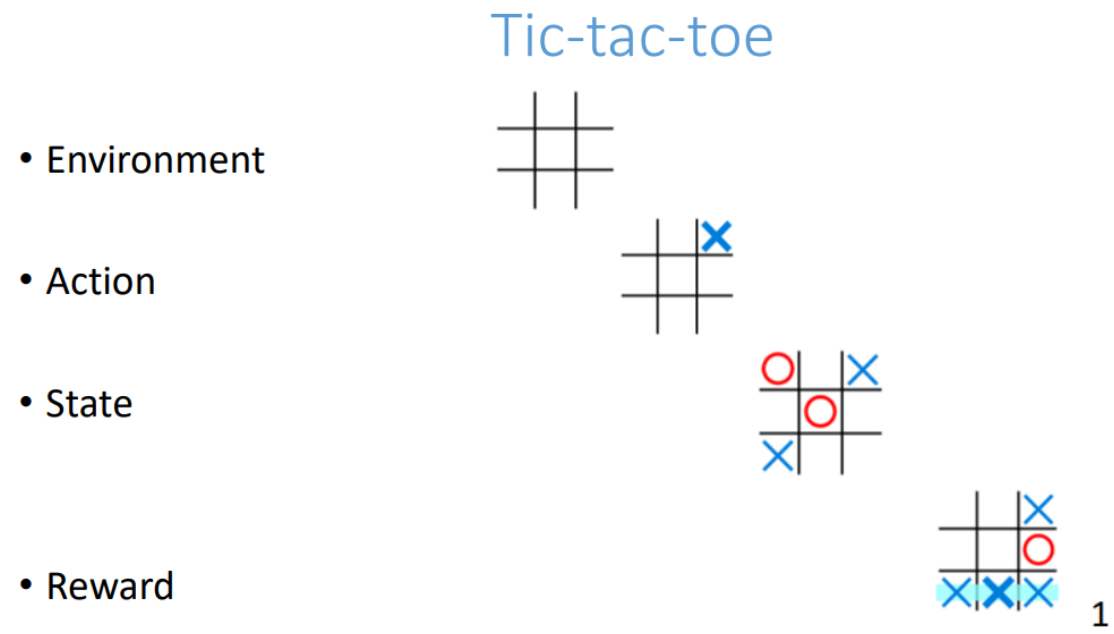
In RL we have states and we map them to some values. These number are obtained by using function estimators.

Deep RL is where we use deep networks as function estimators.

In RL we have interaction between an agent and environment. The environment provides hints to the agents based on actions taken by agent. The agent learns from those hints.

Markov Decision Process

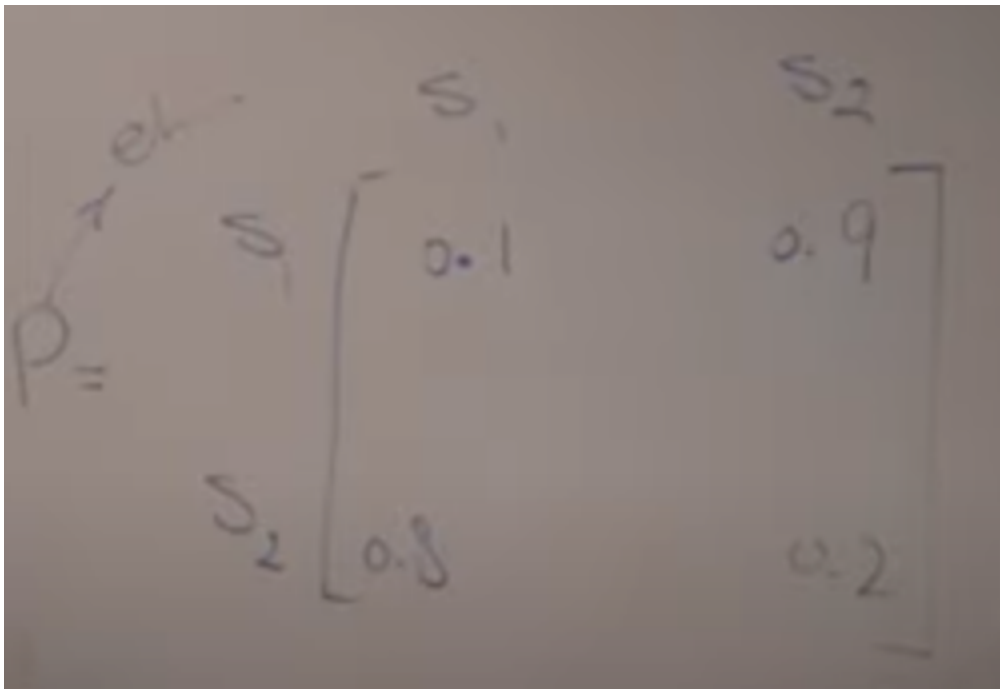
Note: MDP is framework to solve problems. RL is an approach to solve problems and it uses MDP framework to solve problems.



Markov Decision Process (MDP)

- States: $s \in S$
  - The set of all possible situations (s) an agent can encounter.
- Actions: $\mathrm{a} \in A$
  - The set of all possible moves (a) the agent can make.
- Rewards: $\mathrm{r} \in R$
  - Reward model: $\Pr(r_t|s_t, a_t)$
    - Scalar feedback (r) received after executing an action in a state
- Transition model: $\Pr(s_t|s_{t-1}, a_{t-1})$
  - The probability of moving to the next state (s_t) from the current state-action
- Discount factor: $0 \leq \gamma \leq 1$
  - A coefficient ($0 \leq \gamma \leq 1$) determining the present value of future rewards.
  - Discounted: $\gamma < 1$ Undiscounted: $\gamma = 1$

Note: Reward is a function of state and action. Also, reward need not be deterministic, it can be probabilistic.



Transition matrix: It tells us the chances of transitioning from current state to next state by taking a particular action. So every action will have its own transition matrix. This too can be deterministic or probabilistic.

Policy

Policy

- A policy π is a mapping from states to actions,
- $A_t = \pi(S_t)$ (deterministic policy)
- $A_t = \pi(\cdot|S_t)$ (stochastic policy).

Markov Decision Process (MDP)

- **Mathematical Formulation of the Objective:**
  - Find a policy π such that the long-term reward of the agent is maximized.
  - $\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{h} \gamma^t E_\pi[r_t]$

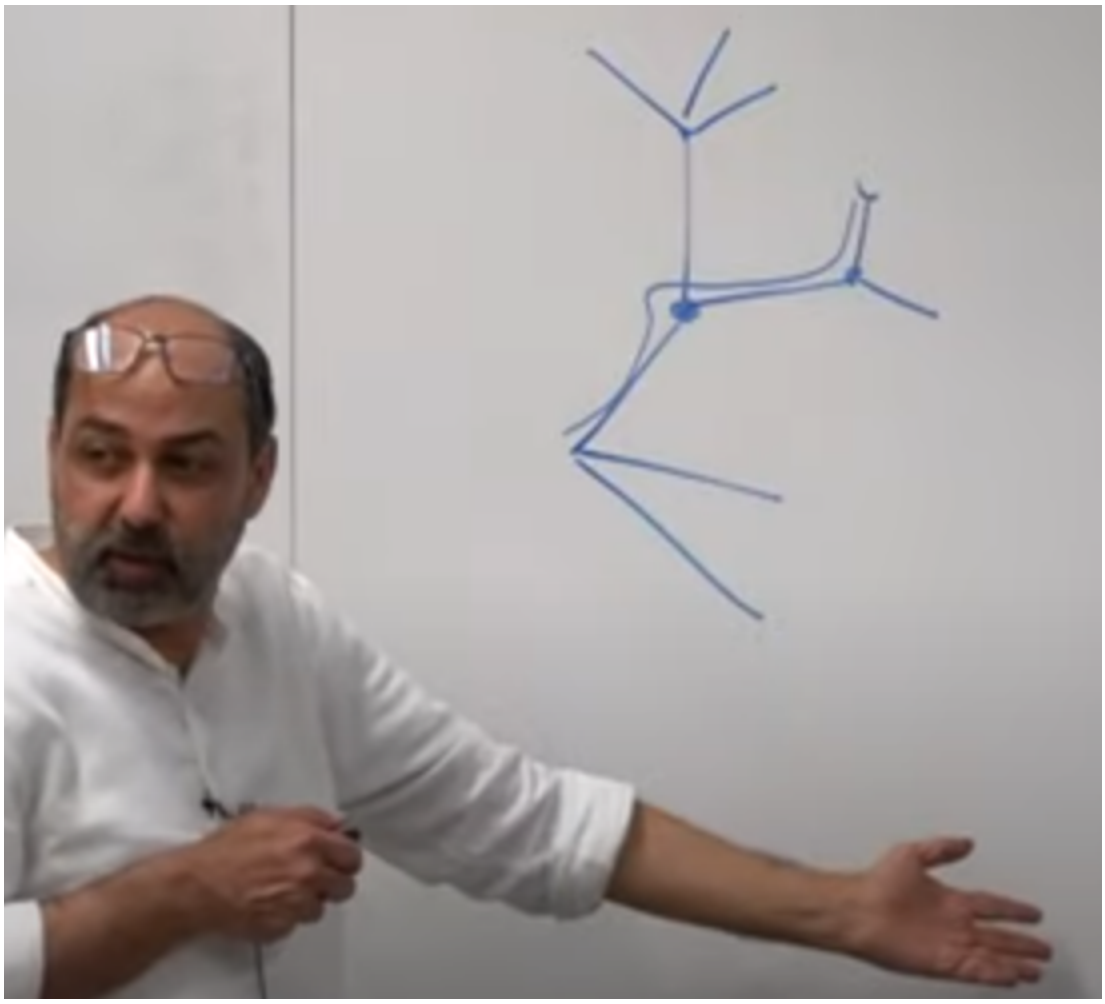
- **Policy:**
  - A policy π dictates the agent's action in each state.
  - The objective is to determine an optimal policy π^* that maximizes expected long-term rewards.

h ⇒ horizon, the number of steps we want to take.

At each time step, total reward = immediate reward + discounted future reward

The optimal policy is decided based on the total discounted reward.

The main problem in Reinforcement Learning is to find this optimal policy.



In the above tree, nodes specify the state at some time step while the edge is transition. We can consider a optimum policy as a path in the graph which gives maximum reward at the end.

Examples of Deterministic and Stochastic Policies

Deterministic Policy:

Scenario: Robot in a maze.

Policy: At a junction, turn right. At a dead-end, turn around.

Characteristic: No randomness, fixed actions.

Stochastic Policy:

Scenario: Marketing strategy for customer interactions (state).

Policy: For a sports product browser, 70% chance to suggest related items, 20% for fitness services, 10% for supplements.

Characteristic: Actions based on probabilities, varied responses.

Bellman's Equation

Bellman's Equation

Definition:

The Bellman equation is a fundamental recursive formula in reinforcement learning that calculates the optimal value of a current state by considering all possible future states. It balances the immediate reward with the maximum expected future rewards.

$$V(s_t) = \max_{a_t} \left[R(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t) V(s_{t+1}) \right]$$

How do we find the optimal policy?

There are two algorithms to obtain optimal policy: Value Iteration and Policy Iteration.

Value Iteration

Value Iteration

$\mathbf{V}^*(s) \leftarrow 0$ **for all** s
repeat until convergence
$\mathbf{V}^*(s) \leftarrow \max_a [R(s,a) + \gamma \sum_{s'} \Pr(s' \mid s,a)\mathbf{V}^*(s')]$

return $\mathbf{V}^*$

$\pi^*(s) = \arg\max_a [R(s,a) + \gamma \sum_{s'} Pr(s'|s,a)V^*(s')]$ for all s.

Note: Optimal policy π^* changes with time. It's non-stationary.

Finds the maximum value for every state. Based on the max value of states, we choose the actions to be taken in every state and hence the optimum policy.

State-Value VS Optimal State-Value

State-Value Function $V^{\pi}(s)$

- Represents the expected total reward for a state s under policy π.

Optimal State-Value Function $V^*(s)$

- The highest expected reward attainable from state s under any policy.

Note: Vpi is a function of state and policy. V* is a function of state only.

Policy Iteration

Policy Iteration

1. **Policy Evaluation:**
   - Update value function based on current policy.
   - $$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s') \quad \forall s$$
2. **Policy Improvement:**
   - Adjust policy based on updated value function.
   - $$\pi(s) \leftarrow \arg\max_{a} \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{\pi}(s') \right] \quad \forall s$$

Here, policy evaluation and improvement happens iteratively. While in Value Iteration, we repeatedly evaluated until convergence and then we found optimal policy.

In terms of complexity, value iteration requires lesser operations on every iteration compared to policy iteration. But policy iteration converges in fewer iterations compared to value iteration i.e. policy iteration converges faster.

Modified Policy Iteration

1. **Partial Policy Evaluation:**
   - Repeat k times:

$$V^{\pi}(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^{\pi}(s') \quad \forall s$$

2. **Policy Improvement:**
   - Update the policy based on evaluated values:

$$\pi(s) \leftarrow \arg\max_{a} \left[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^{\pi}(s') \right] \quad \forall s$$

So we try to mix the two approaches. Instead of repeating till convergence(as in value iteration), we repeat policy evaluation some k number of times and then improve policy. Then, again we repeat k number of times and gain improve policy and so on.

Monte Carlo Estimation

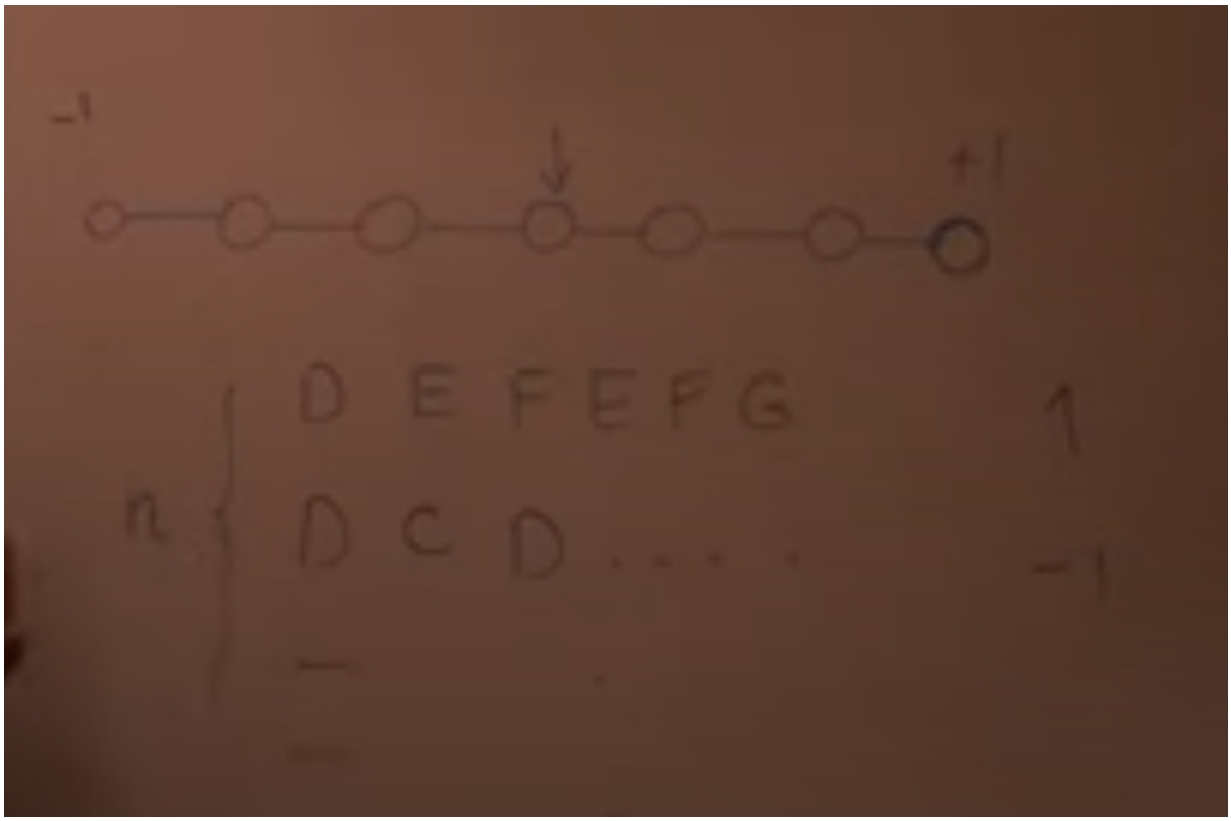
Reinforcement Learning

- Definition
  - States: $s \in S$
  - Actions: $a \in A$
  - Rewards: $r \in R$
  - Transition model: $\Pr(s_t|s_{t-1}, a_{t-1})$
  - Reward model: $\Pr(r_t|s_t, a_t)$
  - Discount factor: $0 \leq \gamma \leq 1$

  - Goal: find optimal policy π^* such that

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{h} \gamma^t E_\pi[r_t]$$

Till now in MDP, we had assumed that we know the Transition model. However, in real world scenarios, we don't. How then do we find state-value function?



In such a scenario, the agent needs to do a random walk in the environment. Let’s say the agent started from state D. The agent will move from one state to another depending on some transition probability which is unknown now. For each walk, the agent will get a final value. At the end of say n walks, we find the average of all these final values. This is our estimate of value for state D.

Monte Carlo Estimation for State-Value Function

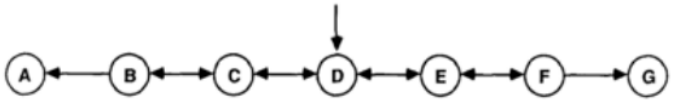
$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t} \gamma^{t} r_{t}\right]$$

$$V^{\pi}(s) \approx \frac{1}{n(s)} \sum_{k=1}^{n(s)} \left[\sum_{t} \gamma^{t} r_{t}^{(k)}\right]$$

- Computes the average return for state s based on sampled episodes.
- $n(s)$: Number of episodes where state s was visited.
- $r_t^{(k)}$: Reward at time t in episode k.

Monte Carlo Estimation for State-Value Function

$$V^{\pi}(s) \approx \frac{1}{n(s)} \sum_{k=1}^{n(s)} \left[\sum_{t} \gamma^{t} r_{t}^{(k)}\right] = \frac{1}{n(s)} \sum_{k=1}^{n(s)} [G_k]$$



Monte Carlo Estimation for State-Value Function

- Let G_k be a one-trajectory Monte Carlo target

$$G_k = \sum_t \gamma^t r_t^{(k)}$$

- Approximate value function

$$\begin{aligned} V_n^\pi(s) &\approx \frac{1}{n(s)} \sum_{k=1}^{n(s)} G_k \\ &= \frac{1}{n(s)} \left(G_{n(s)} + \sum_{k=1}^{n(s)-1} G_k \right) \\ &= \frac{1}{n(s)} \left(G_{n(s)} + (n(s)-1) V_{n-1}^\pi(s) \right) \\ &= V_{n-1}^\pi(s) + \frac{1}{n(s)} \left(G_{n(s)} - V_{n-1}^\pi(s) \right) \end{aligned}$$

- **Incremental update**

$$V_n^\pi(s) \leftarrow V_{n-1}^\pi(s) + \alpha_n (G_n - V_{n-1}^\pi(s))$$

$1/n(s)$

602

Incremental update when we get a new observation in Monte Carlo estimation. For a large n, we see that alpha is very small and hence the State-Value function converges.

Temporal Difference

Incremental update

$$V_n^\pi(s) \leftarrow V_{n-1}^\pi(s) + \alpha_n(G_n - V_{n-1}^\pi(s))$$

Temporal Difference (TD) evaluation

$$V_n^\pi(s) \leftarrow V_{n-1}^\pi(s) + \alpha_n(r + \gamma V_{n-1}^\pi(s') - V_{n-1}^\pi(s))$$

$$V^\pi(s) = E[r|s,\pi(s)] + \gamma \sum_{s'} \Pr(s'|s,\pi(s))\, V^\pi(s')$$
$$\approx r + \gamma V^\pi(s')$$

604

Intuitively

- instead of trying to calculate total future reward, TD simply tries to predict the combination of immediate reward and its own reward prediction at the next moment in time.
- when the next moment comes, the new prediction is compared against what it was expected to be. (temporal difference)
- use this “temporal difference” to adjust the old prediction toward the new prediction.

Temporal Difference Evaluation

Tdevaluation(π, V^π)

- Repeat
  - Execute $\pi(s)$
  - Observe s' and s
  - Update counts: $n(s) \leftarrow n(s) + 1$
  - Learning rate: $\alpha \leftarrow 1/n(s)$
  - Update value: $V^\pi(s) \leftarrow V^\pi(s) + \alpha\big(r + \gamma V^\pi(s') - V^\pi(s)\big)$
    $s \leftarrow s'$
- Until convergence of V^π
- Return V^π

Action-Value Function

Comparison: V^π and Q^π

- **State-Value Function** $V^\pi(s)$:
  - Represents the expected return starting from state s and following policy π thereafter.
  - Defined as: $V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_t = s\right]$
  - Depends only on the current state.
- **Action-Value Function** $Q^\pi(s, a)$:
  - Represents the expected return after taking action a in state s and following policy π thereafter.
  - Defined as: $Q^\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_t = s, A_t = a\right]$
  - Depends on both the current state and the action taken.

For State-Value function, we had to check all possibilities before we get the optimal policy. Its like checking all the possibilities and then deciding the optimal path.

But in case of Action-Value function, we only need to give the action we are taking in a particular state and then it follows the optimal policy.

Bellman's Equation

- Optimal state value function $V^*(s)$

$$V^*(s) = E[r|s,a] + \gamma \sum_{s'} \Pr(s'|s,a) V^*(s')$$

- Optimal state-action value function $Q^*(s,a)$

$$Q^*(s,a) = E[r|s,a] + \gamma \sum_{s'} \Pr(s'|s,a) \max_{a'} Q^*(s',a')$$

where $V^*(s) = \max_a Q \quad (s,a)$

$$\pi^*(s) = \operatorname{argmax}_a Q \quad (s,a)$$

Monte Carlo

- Let G_n^a be a one-trajectory Monte Carlo target

$$G_n^a = \underbrace{r_0^{(n)}}_{a} + \underbrace{\sum_{t=1} \gamma^t r_t^{(n)}}_{\pi}$$

- Alternate between
  - **Policy Evaluation**

$$Q_n^\pi(s,a) \leftarrow Q_{n-1}^\pi(s,a) + \alpha_n \left(G_n^a - Q_{n-1}^\pi(s,a) \right)$$

  - **Policy Improvement**

$$\pi'(s) \leftarrow \operatorname{argmax}_a Q^\pi(s,a)$$

Temporal Difference

- Approximate Q-function:

$$Q\ (s,a) = E\ [r|s,a] + \gamma \sum_{s'} \Pr(s'|s,a) \max_{a'} Q\ (s',a')$$
$$\approx r + \gamma \max_{a'} Q\ (s',a')$$

- **TD**

$$Q_n\ (s,a) \leftarrow Q_{n-1}(s,a) + \alpha_n \left(r + \gamma \max_{a'}(s',a') - Q_{n-1}(s,a) \right)$$

Q-Learning

1. Initialize Q values arbitrarily for all state-action pairs.
2. Repeat until convergence or for a sufficient number of episodes:
   - Choose an action a from state s
   - Take action a, observe reward r and next state s'.
   - Update Q values using the rule:
     $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
   - Update the state: $s \leftarrow s'$
3. Continue with the new state.

Exploration v/s Exploitation

Exploration vs. Exploitation

Exploration:

- **Definition:** Taking random actions to discover new paths and outcomes.
- **Advantage:** Uncovers new possibilities and avoids local optima.
- **Disadvantage:** May lead to immediate suboptimal rewards.

Exploitation:

- **Definition:** Choosing actions that promise the highest reward based on current knowledge.
- **Advantage:** Maximizes short-term rewards and makes use of known information.
- **Disadvantage:** Risks missing out on potentially better options.

Doubts

In Action-State function, is it assumed that current policy is optimum and then via evaluation and improvement it is made better?