RL
UNIT-3

objectives:

1. Is to learn an optimal policy that maximizes the cumulative reward over time through interaction with environment.

primary Types of problems in RL:

2 Types

1. prediction problem. (estimating state values)
   (s)
2. control problem (finding the optimal policy)

Role of γ (discount factor) in RL:

It Determines the importances of future rewards relative to immediate rewards, with values close to 1 indicating future rewards.

episode:

It has a clear beginning and end.

continuing:

It has no defined ending point.

TD-Temporal Difference:

It is a method that combines Monte carlo methods and Dynamic programming, where the agent updates the value func^n based on estimates rather than complete episodes.

value iteration Alg^n:

used to compute the optimal value func^n & policy iteratively by updating the values estimates for each state based on bellman equ^n.

Prediction problem :

It is useful in evaluating how good a particular policy. i.e. once the values are known. It provides insights into the behaviour of the policy & can serve as a foundation for improving policy.

→ Main goal is to evaluate (or) estimate the expected cumulative reward given policy in specific environment.

→ value func^n : The agent is concerned with finding either the

i. state-value func^n — $V\pi(s)$

ii. action-value func^n - $Q\pi(s,a)$ under specific policy π \ pi

i. SVF : Represents expected cumulative reward when starting from state sss & following policy π.

ii. AVF : Represents expected cumulative Reward when starting from state s, taking action a, then follow^n policy π/pi.

Ex:

In gaming environment, Agent could use prediction to evaluate how effective a specific strategy in reaching a high score without trying optimize the score directly. It helps Agent understand the value of different states under a fixed policy.

Techniques:

1. Monte carlo Methods - Estimate value func^n by averaging returns.
2. Temporal - difference (TD) learning - updates the value func^n based on current estimate & a bootstrapped estimate from the next state.

TD methods like TD(0) are widely used for prediction problems in RL.

control problems:

Aim is to find the optimal policy that maximizes the expected cumulative reward.

→ It is the best strategy used in RL problems.

→ Main goal is to optimize the policy so that the agent can make decisions that maximize its long-term reward.

optimal value func^n:

i. optimal state-value func^n – $V*(s)$ → Maximum expected reward starting from state sss & following optimal policy.

ii. optimal Action-value func^n – $Q*(s,a)$ → maximum expected reward starting state sss, taking Action aaa, & following the optimal policy afterward.

policy improvement & iteration:

- It involves PI where Agent updates its policy based on the value func^n.
- P.Iteration & value iteration are two key methods in CP.
  - P.Iteration → It alternates b/w evaluating the current policy & improving it.
  - value iteration → combines Two steps
    1. continuously updating the value func^n towards optimality
    2. deriving the policy from the updated values.

→ CP's has required exploration and exploitation trade off.

- exploration → Agent tries new actions to discover potentially for better policy
- exploitation → Agent uses known information to maximize rewards based on its current knowledge.

Ex:

Agent will explore various paths initially. as it learns which paths lead to higher reward.

(Shortest distances).

Techniques:

1. Q-learning
2. SARSA – State-Action-Reward-Next State-Action.
3. Deep Q-Networks (DQN).

Model-based Algorithm	Model-free Algorithm
It uses a model of environment to predict future states & rewards	It donot use a model and instead learn directly from interactions with the environment like Monte carlo methods

Model-based Algorithm:

uses a model of environment to plan & make decisions.

→ It allows the agent to predict outcomes of actions, improving efficiency & learning speed.

→ A model in RL consists of 2 components.

1. Transition model:

T(s,a) - predicts next state "s'" based on the current state "s" & action "a".

2. Reward Model:

R(s,a) - Predicts the reward "r" for taking action "a" in state "s".

→ By having a model an agent can formulate trajectory without interacting directly with the environments.

trajectory: the sequence of states & actions an agent takes within an environment over a period of time.

components:

1. learning the model:

- somecases, the model may be provided explicitly by the environment eg: board games like chess which has rules fixed.
- othercases, especially in unknown or complex environments the model must be learned from data.
- common techniques for learning models like NN, regression.

2. planning with the model:

- onces model is avaliable agent can use it to plan actions by simulating various scenarios.
- Algorithms typically use DP, MCTS (Monte carlo Tree search) methods.

Algorithms:

Several common approaches & algorithms.

a. DP.

b. MCTS.

c. Dyna-Q.

d. Policy search with model-based planning.

a. Dynamic programming:

- DP methods like policy iteration & value iteration use the model to perform sweeps over the state space & iteratively improve the value funcn & policy.
- It Requires complete & accurate model, so it often used in discrete or fully observable environments.

Disadv: It doesn't scale well for large complex environments

Key concepts:

1. Episode : complete sequence of states, actions & rewards from the start to the terminal.

2. Return : total accumulated reward from time step t+1 onwards in (Gt) episode. It says as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

R_t = reward received at time t.

γ = discount factor

Eg.

→ Suppose we wish to estimate $V_\pi(s)$ the optimal (max) value of a state s under policy π, given a set of episodes obtained by following π and passing through s.

Each occurrences of state s in an episode is called a visit to s. s may be visited multiple times in the same episodes. Let us call the first time s is visited in an episode, the first visit to s. The first visit to MC methods estimates $V_\pi(s)$ as the average of the returns following first visit to s, whereas the every visit MC methods averages the returns following all visits to s.

Types :

1. First - visit MC.
2. ~~Sec~~ Every - visit MC.

. F-VMC :

In this we update the value of state s only when it is encountered for the first time in an episode. After running several episodes we compute the average return for each state from the first time it was visited.

Steps:

For each episode, initialize the episode with the initial state.

ii. Track the first visit to each state.

iii. calculate return for each state s, where s is first time it appears in the episodes.

iv. update the value of state s based on average of the returns observed during the first visit.

$$V(s) = \frac{1}{N(s)} \sum_t G_t$$

$N(s)$ = no. of times state s has been visited for the first time.

2. E-VMC:

In this we update the value of state s every time it visited during an episode, not just first time.

→ This uses occurances of the state to compute the average return.

Steps:

i. For each episode, initialize the episode with the initial state.

ii. For every occurances of each state s in the episode compute the return

iii. update the value of state s by averaging all returns

$$V(s) = \frac{1}{N(s)} \sum_t G_t$$

$N(s)$ = no. of times state s has visited in total.

Algorithm:

1. initialization.

- initialize $V(s)$ for each state sεs
- $N(s) = 0$.

2. For each episode.

a. generates a sequence of states, actions & rewards by following policy π

b. for each state sεs encountered during the episodes.

i. calculate the return G_t

ii. It uting F-VMC update v(s)

iii It uting E-VMC update v(s)

c. update value Estimates.

- It updates after completion of each episode & update the state value v(s) using averaging return.

d. Repeat.

- until process of many episodes to obtain accurate estimates of v(s).

Adv:

1. Model-free
2. Simple to implementation
3. convergence

Disadv:

1. High variance
2. slow convergence.

Applic

1. Robotics
2. Game playing

Example for First-visit & Every-visit MC:

let us consider two episodes they are

1. A+3 → A+2 → B-4 → A+4 → B-3 → terminate
2. B-2 → A+3 → B-3 → terminate.

In above episodes we have 3 states they are A, B & terminate.

→ Here A+3 → A means the transition from state A to A (A → A) with reward = 3

First-visit MC:

for e1.

calculating V(A):

Whenever "A" finds then from there calculate all the value at last.

Hence,

For 1st episode = 3+2+(-4)+4+(-3) = 2

for 2nd episode = 3+(-3) = 0

In 2nd episode A starts at second position to start from second

As we got two terms, we will be averaging these two value

$$V(A) = \frac{2+0}{2} = 1$$

calculating $V(B)$:

for 1st episode $= -4+4+(-3) = -3$

2nd " $= -2+3+(-3) = -2$

Averaging two episodes

$$V(B) = \frac{-3-2}{2} = \frac{-5}{2}$$

Every-visit MC:

taking same example but here Every visit will be counted like

calculate $V(A)$:

1st episode $= (3+2+(-4)+4+(-3)) + (2+(-4)+4+(-3)) + (4+(-3))$

$= 2+(-1)+1$

2nd episode $= 3+(-3) = 0$

Averaging:

$$V(A) = \frac{2+(-1)+1}{4} = \frac{2}{4} = 0.5$$

Similarly

calculating $V(B)$.

1st episode $= (-4+4+(-3)) + (-3) = -3 + -3$

2nd episode $= (-2+3+(-3)) + (-3) = -2 + -3$

Averaging:

$$V(B) = \frac{-3+-3+-2+-3}{4} = \frac{-11}{4} = -2.75$$

Online implementation in MC methods :

The online implementation of MC policy evaluation is a method in RL to estimate the value of a policy by averaging returns from sampled episodes.

-> It is particularly useful in Dynamic, real-time environments where an agent must learn & make decisions continuously. such as robotics, online games etc.

key concepts.

1. value Func^n - calculates based average returns observed after visiting the state

2. Incremental Update - Instead of waiting until the end of a large no. of episodes the value estimate V(s) is updated incrementally with observed return.

3. learning Rate α.

adjustable parameter

Adv:

1. Flexibility
2. Efficiency
3. Adaptability

Disadv:

1. High variance
2. learning rate sensitive

Algorithm :

1. Initialization:

   - Initialize V(s)
   - choose a small fixed learning rate (0,1) to control the update rate.

2. Run an Episode :

   - For each episode

     a. Initialize episode with the starting state S.

     b. Interact with environment following policy π / pi π to generate a sequence of actions, states, reward

3. calculate return Gt for each state.

   formula : $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$

4. update value func^n incrementally.

$$V(s) \leftarrow V(s) + \alpha (G_t - V(s))$$

$G_t - V(s)$ = TD error

5. Repeat