RL - CIA 2

Course	🤖 Reinforcement Learning
Date	@October 31, 2024 4:00 PM → 6:00 PM
Type	Assignment
Status	Completed
Submitted Time	@October 31, 2024 6:36 PM

Table of Contents

Reinforcement Learning and Grid World Implementation

Concepts

- **Agent**: The learner or decision-maker that interacts with the environment.
- **Environment**: The external system with which the agent interacts. Gives feedback as rewards and penalties.

- **State**: A representation of the current situation of the agent within the environment.
- **Action**: A choice made by the agent that affects the state of the environment.
- **Reward:** Positive or Negative, Agent does not know this in prior.

Markov Decision Process (MDP)

An MDP provides a mathematical framework for modeling decision-making situations where outcomes are partly random and partly under the control of a decision-maker.

- A set of states ⇒ S.
- A set of actions ⇒ A .
- A transition function $P(s'|s,a)$ that defines the probability of moving to state s' from state s after taking action a.
- A reward function R(s, a) that provides feedback for taking action a in state s.

Q-Learning

- Used to find the optimal action-selection policy for an agent.
- Uses a Q-table to store values representing the expected utility of taking a given action in a given state.
- The Q-value is updated using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Where:

- α is the learning rate
- r is the reward received after taking action
- γ is the discount factor
- s′ is the new state after taking action a

SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy reinforcement learning algorithm that updates the action-value function based on the current state, the action taken, the reward received, the next state, and the action chosen in that next state. Unlike Q-learning, which uses the maximum Q-value of the next state for updates, SARSA takes into account the action that the agent actually chooses to take in the next state, making it more aligned with the policy being followed.

Update Rule ⇒

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Grid World Implementation

- Implemented a 100×100 World
- Ensure that there is a valid path using A* heuristic
- Visualize the plot

Algorithm - Q Learning

Hyperparameters ⇒

- **Learning Rate (α)**: 0.1
  - The agent will update its Q-values by 10% of the difference between the target and current Q-value.
- **Discount Factor (γ)**: 0.99
  - Future rewards are considered almost as important as immediate rewards (Can be tuned)
- **Exploration Rate (ϵ)**: 0.1 (initially)
  - Controls the exploration vs. exploitation trade-off

Initial Q-values Table ⇒

- The Q-values for each state-action pair are initialized to zero:

```
self.Q_values = defaultdict(lambda: defaultdict(float))
```

- For every state-action pair, Q(s, a) starts at 0.

Steps

1. Initialize Q-values: Create a dictionary to store Q-values for each state-action pair (Initially 0):

   ```
   self.Q_values = defaultdict(lambda: defaultdict(float))
   ```

2. Set Hyperparameters
3. For each episode (total episodes = 1000) → Set the initial state to the starting position of the agent (`self.environment.start`).
4. For each step within an episode (up to a maximum number of steps per episode):
   - Choose an action a based on the current state s using epsilon-greedy strategy:
     - With probability ϵ, select a random action (exploration).
     - With probability $1 - \epsilon$, select the action with the highest Q-value for the current state (exploitation).
5. Take Action and Observe Reward:
   - Observe the next state s′ and receive reward from `self.rewards[next_state]`.
6. Update Q-value: Using above formula
7. Transition to Next State
8. Repeat Steps 4-7 until reaching a terminal state (goal) or exceeding a maximum number of steps per episode.
9. Decay Exploration Rate: encourage more exploitation as learning progresses
10. End Episode

Comparing with DP

Similar to Q-Learning, we create a DP Class and Compare it with the Q-Learning to find the optimal policy.