Reinforcement Learning Based on Monte Carlo Method

Problem:

Here we will use techniques based on Monte Carlo (MC) estimators to solve reinforcement learning problems in which we don't know the environmental behavior. Here the algorithm based on MC will learn based on an episode by episode strategy and estimate the state-action values over many episodes to find an optimal/good policy employing First VisitMonte Carlo Control. Here we will use similar approach to implementing the MC but on our FrozenLake8x8 (both slippery and none slippery) and shows the performance of these policies.

Proposed Solution:

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Using the gym toolkit and numpy library, the parameters:

- env is the initialized Open AI gym environment object with the FrozenLake8x8-v0 and FrozenLake8x8NotSlippery-v0 in the two programs.
- S is the agent's state

- V is a vector, the value to be uses for estimation
- Gamma is the discount factor
- Policy is a matrix where each cell has the probability of performing an action a in state s.
- Theta is the threshold

The program employs a value iteration algorithm that computes the optimal state value function by iteratively improving the estimate of V(s).

Following the prescribed algorithm,

- def calc_values(env, s, V, gamma) – is used to calculate the state-value. It returns a vector containing the estimated value of each action. It computes the values by:

```
action_values[action] = action_values[action] + probability * (reward +
discount_factor_gamma * V[next_state])
```

- def create_policy(env, gamma=1.0, theta=0.001) – is employed to create a policy after evaluation. It selects the best action, updates the vector (value function) if there is a change in value and updates the state and policy for optimum performance.

- def apply_policy(env, n_episodes, policy) – This function applies the updated policy to the real environment by implementing the best actions possible in the current state, executing an episode. It keeps tabs of the goals scored and displays (renders) the steps taken in taken in/for execution of an ongoing episode. It keeps tabs of the goals scored and number of episodes executed in total for final efficiency evaluation.