AE598 : Reinforcement Learning Assignment 2

Girish Madhavan Venkataramani
University of Illinois, Urbana-Champaign
Champaign, Illinois
gmv4@illinois.edu

***Abstract*—This report explores model based Reinforcement Learning algorithms like policy and value iteration for an agent in frozen lake environment**

I. Introduction

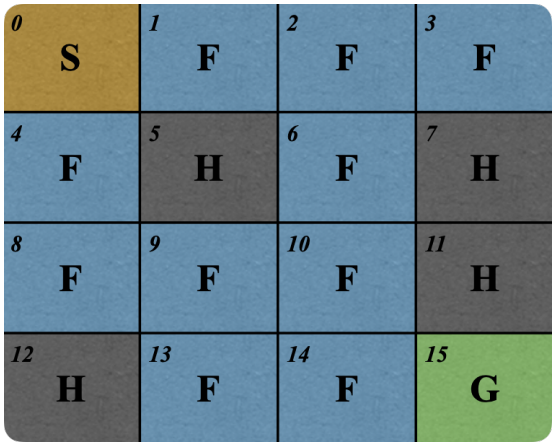

Fig. 1. State space

The task of this assignment is to implement policy and value iterations as detailed in Sutton & Barto's Reinforcement Learning book (Chapters 4.3 & 4.4) on the above version of frozen lake environment. In 1 the states are as follows :

- S : Start
- F : Frozen
- H : Hole
- G : Goal

The numerical value of the state is obtained as follows : *curr row* $\times$ *num cols* + *curr col*.

Moreover, if the environment is slippery then the agent moves in the direction of the intended action with probability $\frac{1}{3}$ and it could move in the direction perpendicular to that of intended action with probability $\frac{1}{3}$ for each of the two perpendicular directions. However if the environment is not slippery then the agent moves in the direction of the intended action with probability 1

Now the actions space is defined as follows along with their array indices used in "actions" dictionary from code :

0 "UP" : [0,-1]
1 "DOWN" : [0,1]
2 "RIGHT" : [1,0]
3 "LEFT" : [-1,0]

In the case that the agent is in the boundary states (adjacent to any wall), actions that tell the agent to go through the wall will cause the agent to stay in place.

A reward of $+1$ is given for any transitions that lead to Goal [15] state from surrounding states, any transitions that reach a Hole [5,7,11,12] are given 0 reward, and any transitions leading to frozen locations are given a reward of 0. Termination occurs when the agent reaches Goal or any of the Hole states. A discount factor of $\gamma = 0.95$ is used for value computation.

II. Method

The goal of this homework is to obtain the optimal policy for the frozen lake environment when the environment model is unknown (i.e $P(s', r|s, a)$ is not available) preventing us from building the value or policy table using policy or value iteration methods. So now we use sampling based algorithms to find the optimal policy, before going through those algorithms a brief overview of the tools that will be used in them (GLIE and Alpha Scheduling) are defined as follows :

A. GLIE Policy (for ϵ-scheduling)

GLIE (Greedy in the Limit with Infinite Exploration) ensures convergence. For an ϵ-greedy policy, this requires that ϵ decays such that:

1) All state-action pairs are visited infinitely often.
2) The policy converges to the greedy policy: $\lim_{k\to\infty} \pi_k(a|s) = \mathbb{I}(a = \arg\max_{a'} Q_k(s, a'))$.

B. Alpha (α) Scheduling

The learning rate α_t for each state-action pair must decrease over time to ensure stable convergence.
A and B are implemented as follows :

$$m_t = \max(m_{min}, m_{decay} \cdot m_t), \quad m_t = \alpha_t \text{ or } \epsilon_t$$

The algorithms discussed are based on the foundational text by Sutton and Barto [1], covering sampling based methods :

C. First-Visit Monte Carlo (MC) Control

The return from time step t is the total discounted reward:

$$G_t = \sum_{k=0}^{T-1-t} \gamma^k R_{t+k+1}$$

The action-value function $Q(S_t, A_t)$ is updated using the observed return G_t:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)]$$

Algorithm 1 First-Visit MC Control

Initialize for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary (e.g., 0)
  $\pi(a|s) \leftarrow$ arbitrary, e.g., $1/|\mathcal{A}(s)|$
  $\texttt{Returns}(s,a) \leftarrow$ an empty list
Initialize $\epsilon_t = 1, \epsilon_{decay} = 0.999999, \epsilon_{min} = 0.0001$
Initialize $\alpha_t = 0.1, \alpha_{decay} = 0.99999, \alpha_{min} = 0.0001$

loop
  Generate an episode following π:
  $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if (S_t, A_t) is visited for the first time **then**
      Append G to $\texttt{Returns}(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(\texttt{Returns}(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ ▷ The greedy action
      for all $a \in \mathcal{A}(S_t)$ **do**
        if $a = A^*$ **then**
          $\pi(a|S_t) \leftarrow 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)|$
        else
          $\pi(a|S_t) \leftarrow \epsilon/|\mathcal{A}(S_t)|$
  $\epsilon_t \leftarrow \max(\epsilon_{min}, \epsilon_{decay} \cdot \epsilon_t)$
  $\alpha_t \leftarrow \max(\alpha_{min}, \alpha_{decay} \cdot \alpha_t)$

This algorithm has only sampling and offers no bootstrapping hence its computationally very intensive and its on policy as well.

D. SARSA (On-Policy TD Control)

The on-policy TD update uses the action, A_{t+1}, that is actually taken in the next state:

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

Algorithm 2 SARSA: On-Policy TD Control

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Initialize $\epsilon_t = 1, \epsilon_{decay} = 0.999999, \epsilon_{min} = 0.0001$
Initialize $\alpha_t = 0.1, \alpha_{decay} = 0.99999, \alpha_{min} = 0.0001$
for each episode **do**
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  loop for each step of episode
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S', A') - Q(S,A)]$
    $S \leftarrow S'; A \leftarrow A'$
    break if S is terminal
  $\epsilon_t \leftarrow \max(\epsilon_{min}, \epsilon_{decay} \cdot \epsilon_t)$
  $\alpha_t \leftarrow \max(\alpha_{min}, \alpha_{decay} \cdot \alpha_t)$

This algorithm is on policy and does bootstrapping on the obtained samples to make updates, therefore its not as demanding as Monte Carlo but convergence could take a bit longer.

E. Q-Learning (Off-Policy TD Control)

The off-policy TD update uses the value of the best possible action, $\max_a Q(S_{t+1}, a)$, from the next state:

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

Algorithm 3 Q-Learning: Off-Policy TD Control

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Initialize $\epsilon_t = 1, \epsilon_{decay} = 0.999999, \epsilon_{min} = 0.0001$
Initialize $\alpha_t = 0.2, \alpha_{decay} = 0.99999, \alpha_{min} = 0.0001$
for each episode **do**
  Initialize S
  loop for each step of episode
    Choose A from S using policy derived from Q
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S,A)]$
    $S \leftarrow S'$
    break if S is terminal
  $\epsilon_t \leftarrow \max(\epsilon_{min}, \epsilon_{decay} \cdot \epsilon_t)$
  $\alpha_t \leftarrow \max(\alpha_{min}, \alpha_{decay} \cdot \alpha_t)$

Finally with Q-Learning, it performs bootstrapping for updates and is off-policy. Since it uses the greedy policy to make its updates it converges faster, but if not careful it could suffer from maximization bias.

III. Results

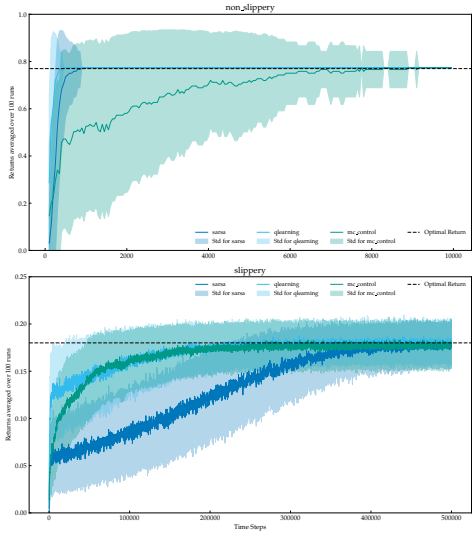

Fig. 2. Evaluation Return Plots

Greedy policy for different algorithms are shown below :

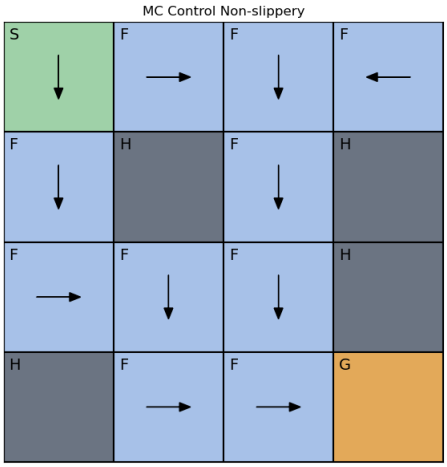

Fig. 3. Monte Carlo Control - No slip

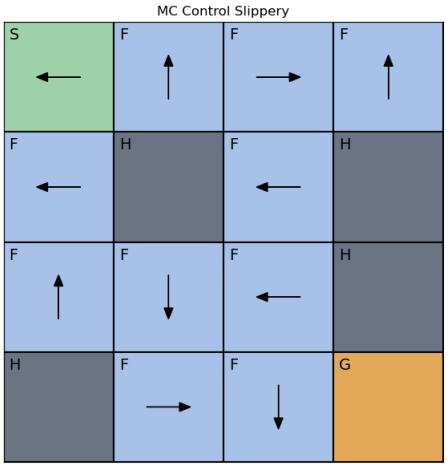

Fig. 6. Monte Carlo Control - slip

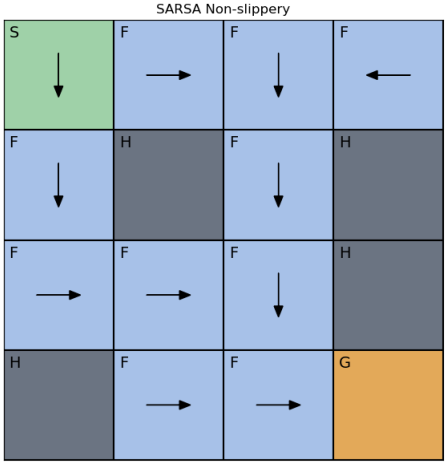

Fig. 4. SARSA - No slip

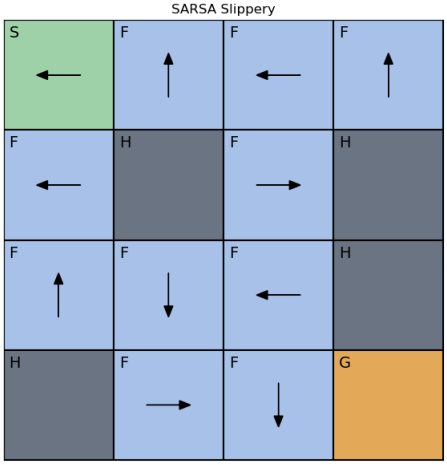

Fig. 7. SARSA - slip

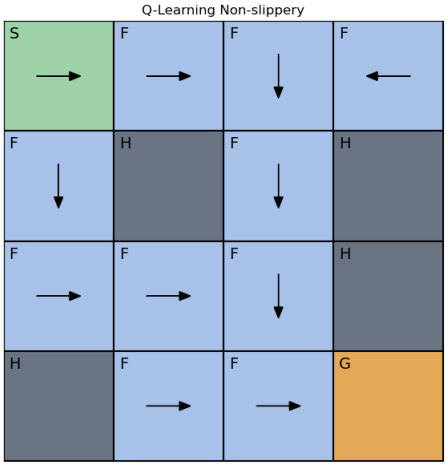

Fig. 5. Q-Learning - No slip

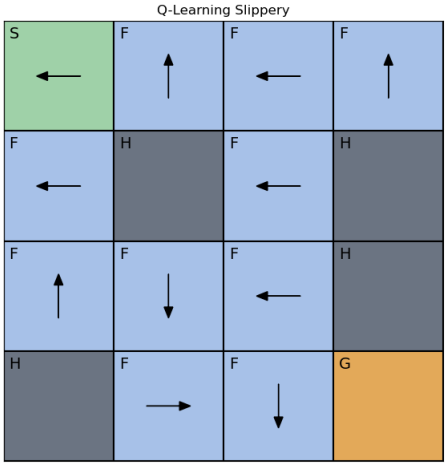

Fig. 8. Q-Learning - slip

IV. Conclusion

- From 2 we can observe that Q-learning often trains faster in both cases (slippery and non slippery) thanks to the carefully chosen set of hyper-parameters, which prevent it from getting swayed by maximization bias.
- The tradeoff between bias and variance is visible in 2 for the first visit to Monte Carlo, as it has a significantly larger standard deviation compared to other scenarios.
- Moreover in both cases, the values to which these plots converge is the same as $V^*(0)$ (computed using value iteration in previous assignment, 0.18 for slippery and 0.77 for non slip), this shows that through sampling and bootstrapping techniques we can converge to the optimal value or policy table when going model-free. Apart from these, 3 to 8 have converged to their matching counterparts from assignment 1, with some exceptions in certain states like 7 where going left or right does not matter and has the same final action.
- One reason why Q learning outperforms other methods here could be due to the sparse reward distribution and on-policy methods taking significant time to converge in the slippery case because of the environment being highly stochastic and the policy starting off as more exploratory. As Q learning updates its Q table using greedy action, it is able to converge slightly faster. Monte Carlo's lower frequency of policy improvement slows down on converging to the optimal policy.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018.