Project: 2

1.) Approach:

a.) Background:

In this project, we implement 2 TD methods: SARSA and Q-Learning. The environment used is "Frozen Lake - V0" from the OpenAI Gym library.

i.) The SARSA algorithm given in our textbook is as follows:

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Similar to most of the On-policy methods, we estimate the Q-values keeping the target and behavior policy as the same, in the implementation, we keep the policy to be epsilon-greedy as given in the example. Now, theoretically SARSA converges to the optimal policy if all the state-action pairs are visited infinite times and then only the policy converges with full guarantee. However, in practical cases the convergence of the policy depends on the environment complexity, the agent meta-parameters, etc.

Online learning methods such as SARSA don't get stuck in an episode loop as they quickly learn that such policies are bad and with clever reward settings, it'll push SARSA to another one.

ii.) The Q-Learning algorithm given in our textbook is as follows:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Now, unlike SARSA, Q-learning is an Off-policy TD learning method where the target policy and the behavior policy are different. In other words, it approximates the q-values independent of the policy being followed (i.e., the update equation follows the maxima under the actions operation whereas the action and state pair can be retrieved by epsilon- greedy approach). The convergence criteria is similar to that of SARSA.

b.) **Implementation**:

We implement the *Value Iteration* algorithm as explained above for the *"FrozenLake-v0"* environment on OpenAI-gym. In this project, we turn the (is_slippery) hidden Markov transitions off to focus on the basic algorithm used. The environment is grid-based and can be visualized as follows:

State Space			
0	1	2	3
4	5	6	7
8	9	10	11
11	12	13	14

Action Space	
LEFT	0
DOWN	1
RIGHT	2
UP	3

Each of the state space cells can have the value H/F/G/S indicating a "hole", "frozen section", "goal", or "Start" respectively. The agent can take 4 actions and will transition accordingly, hitting the walls brings it back to the same cell with 0 rewards. The only transition that results in a +1 reward is when it reaches the cell *"G"*.

c.) **Algorithm:**

Below, we define the pseudo-code for our implementation, it is granular enough and can be extended to other similar grid-world problems.

Pseudo-code: SARSA

Inputs:

- The frozen lake environment.
- Discount factor (γ) …(0.9 value was used for the experiments)
- Learning rate (α)
- Exploration constant (ε)

Returns:

- State value function ~*Reward (R)*

Initialize: *V(s), v(s)* arrays with the size = $|S|$
Q(s, a) array with size = $|S|$ x $|A(s)|$ ….(*A(s)* in this case is = 4 for all the states)

while (*iteration* ≤ *Maximum allowed episodes*):
Initialize '*s'* and select *'a'* according to the greedy policy using the *Q-values, t=0*
While *s* in not a terminal state:
| Collect s_{t+1}, r_{t+1}, *and terminal indicator.*
| Select a_{t+1} according to the s_{t+1} state
| $Q(s_t , a_t)$ += α*[r_{t+1} + γ$Q(s_{t+1} , a_{t+1})$ − $Q(s_t , a_t)$]
| $t = t + 1$
| *Append* r_{t+1} *to Rewards List for further analysis*
return *Q(s, a), Rewards List*

The pseudo code for Q learning is very similar to the above except this time we use a different policy to select the action and a different one to update. Selection can be done by same greedy epsilon policy but the update looks like this:

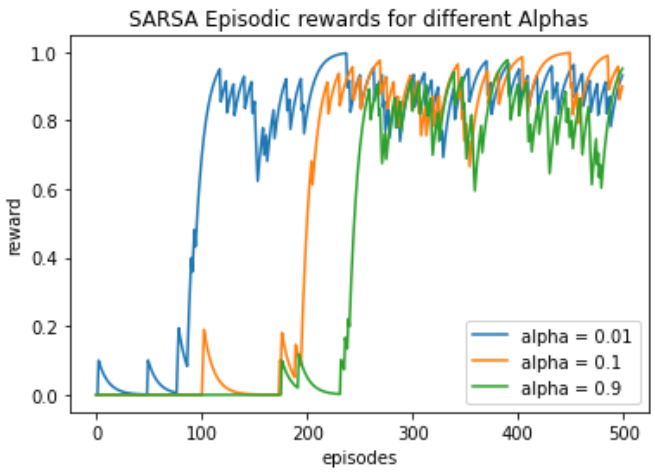
$$Q(s_t , a_t) \mathrel{+}= \alpha * [\, r_{t+1} + \gamma * \max_a Q(s_{t+1} , *) - Q(s_t , a_t)]$$

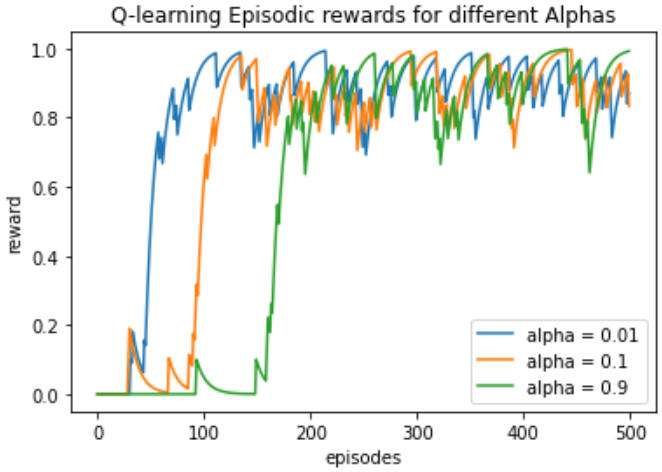
2.) **Experimental Results:**

Extensive experimentation has been carried out to determine the effect of the learning rate, exploration constant, and other general differences between SARSA and Q-Learning. Additionally, the board chosen was of the default category.

i.) Learning Rate:

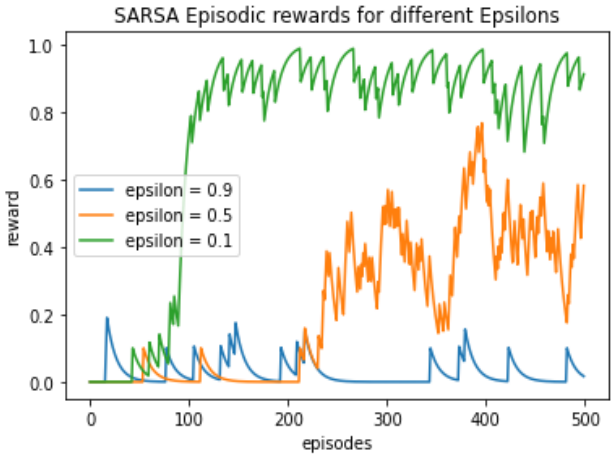
Effect of the learning rate on performance (average testing reward) with exploration of 0.1 that has been modified to follow the moving average with a weight of 0.9 for better clarity, shows that a lower learning rate leads to better performance in both the TD methods:

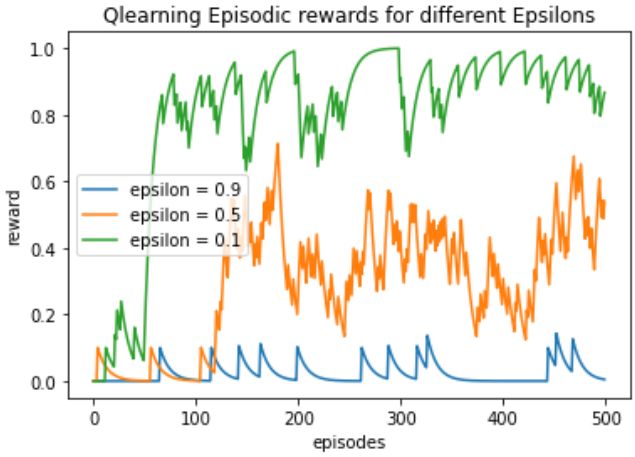




ii.) Exploration rate:

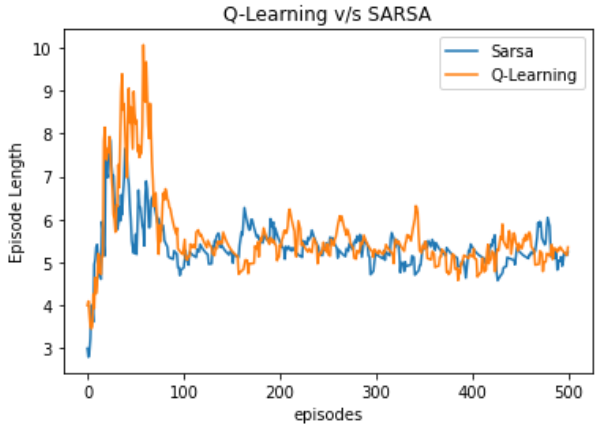
The exploration rate of 0.1, 0.5 and 0.9 has been used indicating low to high exploration preference in the given order. It seems like this also has a similar effect on SARSA and Q-Learning methods with higher epsilon yielding lower rewards. Here are the results:





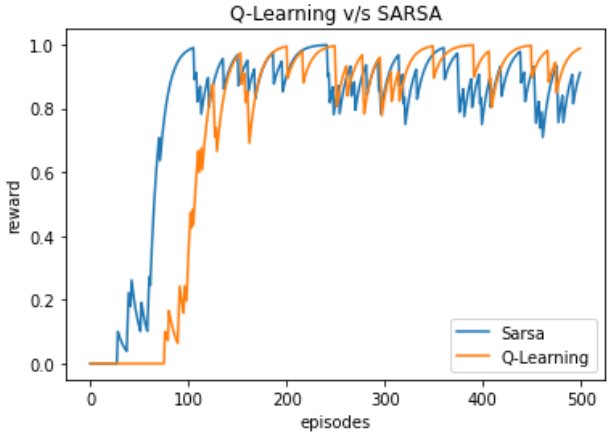
iii.) Episode length before termination.

This is the measure of how initially the agent's judgement is not accurate and it has to wander around, over time we see that it's able to find the Goal position relatively faster:

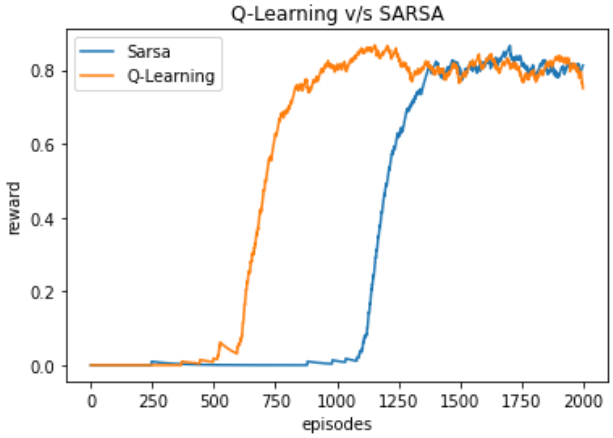


iv.) SARSA v/s Q-Learning.

Unfortunately, the comparison graph between both did not reveal any big distinctions. This is because the problem is simple and convergence is easier, thus the difference between the algorithms is not that clear and follow very close paths with SARSA having a leading trail (in a few results Q-learning also had a leading trail, I suspect initial conditions when the q values are all zeros makes this random to some extent). To explore further, I added the 8x8 board to the environment and then set the parameters as [learning rate = 0.01, epsilon = 0.2, smoothing weights = 0.99] and we have the 8x8's result to the right and 4x4 to the left.



4x4 Frozen Lake Board



8x8 Frozen Lake board

1.) **Discussion**:

The key take-aways here would be the following:

- Lower learning rate can be beneficial for stable learning, very high rates (>1) occasionally result in bad updates and degrading performance.
- Exploration could be set to a higher rate if the state space is extremely large w.r.t., the action space, I found this by experimenting the effects of the same in the 8x8 section where the state-space is much larger.
- Q-learning, due to its greedy Off-policy updates can find routes to maximize the rewards in the initial stages very quickly. However, when multiple high reward state-action pairs might be present then Q-learning will miss on that mainly due to its inherent greedy update.
- Advanced methods exist to control the annealing/exploration rate through a function such that initially the agent explores and after some time focusses on exploiting the known paths/tricks.
- The path length or the episode size would usually reduce with time as the agent learns more and doesn't have to rely explicitly on the exploration to find useful states. This could be false if the reward is set in a way that the longer the episode the more the final rewards. Thus cleverly choosing the reward set-up will greatly affect the performance of these algorithms.

Advantages and disadvantages:

- In multiple reward setups, SARSA might perform better than Q-learning due to the epsilon greedy method that can help SARSA explore.
- Both methods are model-free methods and don't require explicit knowledge of the environment model beforehand and can work with samples and multiple episodes to learn better policies.
- SARSA could be modified to become expected SARSA such that the Expected values can be used, however, this is much more time consuming and computationally complex due to the calculation of the expectations.
- In environments with higher complexity, it is generally observed that SARSA usually preforms better during the saturation of rewards than Q-learning.