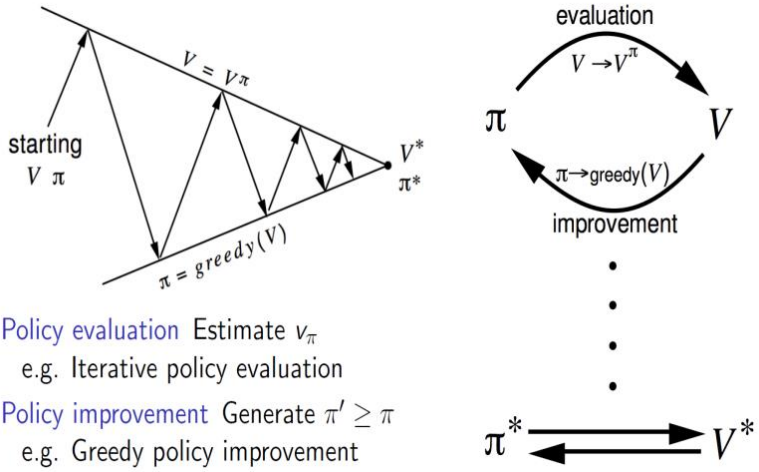
Project Assignment #2: SARSA & Q-Learning on FrozenLake

Approaches:

In this project assignment, the FrozenLake-v1 environment from the OpenAI Gym Python library is used to determine the optimal policy, utilizing the SARSA and Q-Learning algorithms. These algorithms acquire optimal Q-values for every state-action pair within the environment, from which the optimal policy is derived. Both algorithms use a technique of alternating between random exploration and applying the "greedy" policy of maximum value to optimize the final policy.



During the initial 25 episodes, the agent employed random actions to explore the environment. Following this exploration phase, the agent's behavior is governed by an epsilon value that indicates a 90% probability of continued exploration and a 10% probability of utilizing the greedy policy. In the original testing phase comprising 200 episodes, it was determined that after 100 episodes, the probability of exploration would gradually decline to allow for exploitation of the agent's learned information.

For determining the Q-values in the SARSA agent, the following equation was used:

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

For determining the Q-values in the Q-Learning agent, the following equation was used:

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

, where Q is our Q-value matrix, S is our state, A is our action, α is our learning rate, and γ is our discount factor.

Experimental Results:

For the SARSA agent, it was discovered that 200 episodes wasn't enough to get consistent results. Every couple of runs the optimal policy would not result in the agent getting to the present. This is probably due to the lack of exploitation that the agent has after its exploration phase. The number of episodes was moved to 1000 and had a consistent success rate. Below a few of the optimal policies extracted. Each policy is slightly different, resulting in different paths to the same goal taking routes shown below.

Policy: [1. 2. 1. 0. 1. 2. 1. 1. 2. 2. 1. 2. 1. 2. 2. 0.]

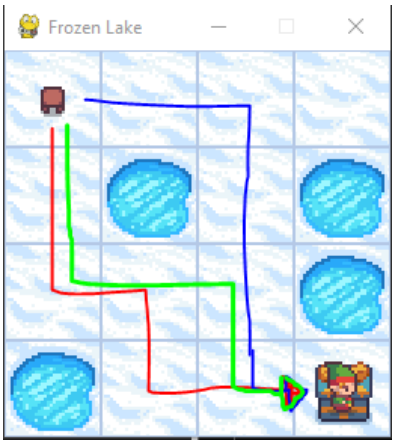
Policy: [1. 2. 1. 0. 1. 0. 1. 0. 2. 2. 1. 2. 2. 2. 2. 1.]

Policy: [2. 2. 1. 0. 1. 3. 1. 2. 2. 1. 1. 1. 2. 2. 2. 0.]

Policy: [1. 2. 1. 0. 1. 1. 1. 1. 2. 2. 1. 3. 2. 2. 2. 1.]

Policy: [2. 2. 1. 0. 1. 1. 1. 1. 2. 2. 1. 0. 1. 2. 2. 2.]

Policy: [1. 2. 1. 0. 1. 3. 1. 1. 2. 2. 1. 2. 1. 2. 2. 0.]



The Q-Learning agent faced difficulties in identifying the optimal policy. I attempted to enhance the learning rate by increasing the number of episodes, but it did not yield positive results. I also experimented with the epsilon variables as described in the approach section with no success in that regard. Despite these challenges, there were a few runs that produced successful outcomes, where the agent was able to navigate through the environment and arrive at the present, however not consistently.

Average reward over 200 episodes: 0.525

Policy: [2. 2. 1. 0. 3. 3. 1. 2. 2. 2. 1. 3. 0. 3. 2. 3.]

Discussion:

In conclusion, this project successfully implemented one temporal-based Reinforcement Learning algorithm SARSA and was not as successful with Q-Learning. SARSA proved to be particularly effective in identifying an optimal policy, while Q-Learning did not perform as well. Although the Q-Value implementation was correctly applied to both objects, as outlined in the approach section, the exact issue remains unclear. It is possible that the problem lies in not fully exploring all possible actions in each state. One advantage of using the SARSA agent that I noticed, is that it is capable of seeing long term rewards rather than immediate ones. Having the ability to see states and actions ahead is a feature that made for a great policy.