SARSA Notes
Wenda Li

For SARSA, I used optimistic initialization for the algorithm. In particular, I take the expected reward for the initialized value. Also, I tune the epsilon greedy's epsilon to be 0.4, after some experiments.

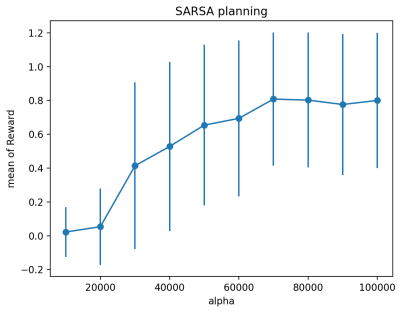
4x4 FrozenLake env behaves well, since the action-state space is quite small.

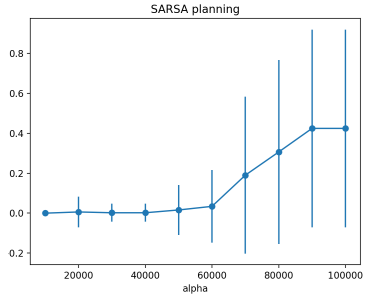
I observed that during initial deployment, Q matrix for 8x8 FrozenLake env when it is initialized, most of the entries are much unexplored. Therefore for the lower number of episodes(~10000-15000), the performance is quite bad. However, when the episodes rises to about 80000, the performance then becomes comparable to that of VI.

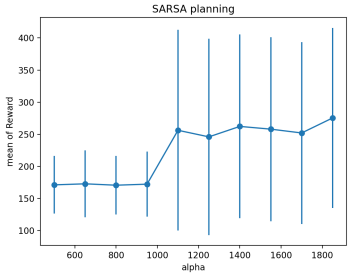
The reason behind this, I guess, is the explorations are more diverse in higher number of episodes. However, it is extremely slow to run high number of episodes, and I found that in fact, most of the troubles with the low learning time is *not* how individual values in Q matrix gets improved to optimal, however with the *unexplored* slots in Q matrix. Therefore a distributed way for SARSA to deal with Q function and policy maybe is a more reasonable option.

Some implementations when I tried to reduce the learning time:

A) SARSA is allowed to return whenever the policy and Q function converges
B) I put policy into the SARSA loop. This allows SARSA to optimize Q function along with policy.
C) For cart pole, I tried about 100-5500 episodes, as it seems to be the reasonable number of episodes to run







Frozen 4x4 **Frozen 8x8** **Cartpole**

From the result one can see that the standard deviation is quite huge, and when the number of steps to get the convergence gets decreased, the standard deviation and average reward is even worse.(see plots/8x8_0.9.png)

Task 2.

To facilitate the learning speed, I cut down the value iterations to about 5000 steps; further, I record the average quality by an array of size t%20, and record the reward of each run. To get mean and standard deviations, I then take the std and mean of aforementioned array, up to step t. Since the initial exploration will generally give 0 rewards, and will be counted in later steps, the

“true reward” of later steps will be bigger. In fact, I observe that when t>800, both algorithm will give about 0.8 in terms of average reward in the consecutive 5 runs.

Onpi works very well, however slow. RMax however, has smaller std and in fact runs much faster than Onpi.