Frozen Lake MDP

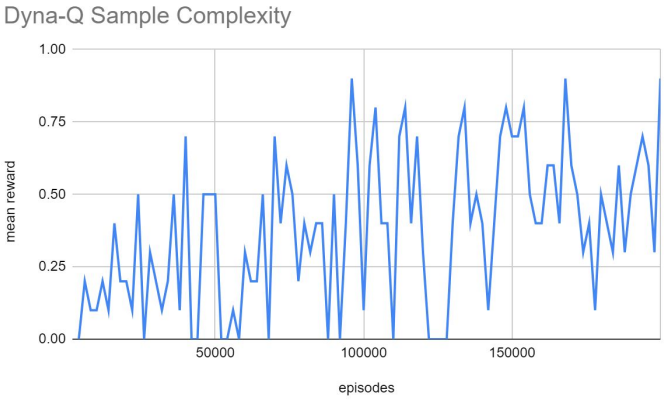
Aaron Barnett
acba242@uky.edu

Abstract

This paper covers three strategies for solving the Frozen Lake environment offered by OpenAI Gym. Dyna-Q, Q-Learning, and SARSA are used and learning rate is assessed through sample complexity.

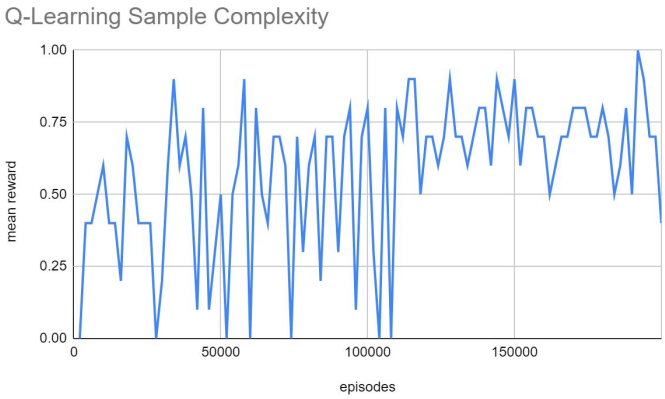
Dyna-Q

To implement Dyna-Q, a 16x4 matrix was used to represent the Q policy, a 16x4x16 matrix was used to represent the transition model, and a 16x1 matrix was used to represent the reward model. Each time a state was visited, a count was incremented within the transition model that corresponded with the previous state, action taken, and the state visited from that transition. Transition probabilities were generated by summing over the transition model matrix. Transition history was stored using a list of tuples representing the state and action taken in that state. The Q-planning step used five planning steps per episode. For all agents, action selection during training used an epsilon greedy policy with a starting epsilon of 0.9 which decayed by 0.0000075 per episode. The maximum between the decayed epsilon and 0.01 was used so epsilon would never be zero. An alpha learning rate of 0.4 and a gamma reward discount of 0.99 was used for all agents. The agent was trained for 200,000 episodes with a maximum 100 time steps per episode. The agent was frozen and tested every 2000 episodes. Testing occurred over 10 episodes with a maximum 100 time steps per episode. The agent greedily followed the Q policy for testing. The mean reward was sampled from the testing phase.



Q-Learning

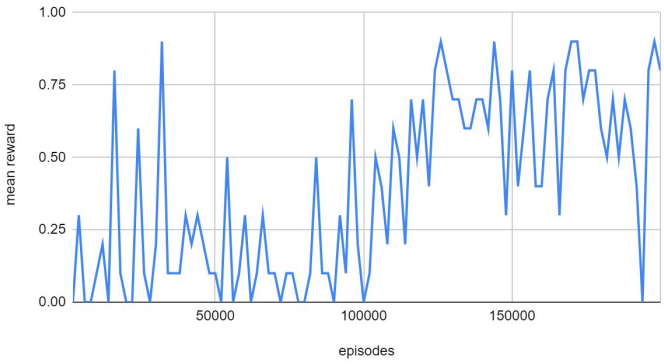
For the Q-learning agent, all hyper parameters, action selection, and decay schedules were the same as the Dyna-Q implementation. A Q policy represented as a 16x4 matrix was used. Q-Learning was tested identically to the Dyna-Q agent. It seems Q-Learning was able to converge slightly before Dyna-Q. Perhaps implementing a schedule to increase planning steps as episodes increase would cause Dyna-Q to converge quicker. Q-Learning also seemed to have better stability once it reached convergence so there could be issues with my implementation of Dyna-Q.



SARSA

For the SARSA agent, all hyper parameters, action selection, and decay schedules were the same as the Dyna-Q implementation. A Q policy represented as a 16x4 matrix was used. SARSA was tested identically to the Dyna-Q agent. SARSA seemed to learn its policy in a much safer way, taking more time to converge, but also arriving at a similar stability as Q-Learning. Both Q-Learning and SARSA outperformed Dyna-Q when considering stability. Perhaps further changes to the alpha learning rate could yield more stability and faster convergence from all agents. All agents seem to converge at around 100,000-125,000 episodes.

SARSA Sample Complexity



Shortcomings

I'm disappointed with how my Dyna-Q agent performed. I believe it should've had better stability and converged quicker than the other two strategies. Given more time, I would implement a schedule to change the amount of planning steps as episodes increase. Changes to the Dyna-Q agent were also difficult because of run time. One run of my implementation could take 30 minutes or more. I would attempt to make speed optimizations if I were to re-write the code so different hyperparameters and planning schedules could be tested easily.