Mitchell Scott
misc4432@colorado.edu

Exploration and Exploitation

May 6th 2020

Background

Over the past year I have been gently immersing myself into the world of machine learning. I started with simple python regressions and classifiers, then I used computer vision and deep learning to program a maze solving robot. Towards the end of this semester my teammates introduced me to reinforcement learning which I ambitiously pursued. I decided to start with a simple environment so I chose CartPole-v0. Soon after implementing a jumbled mess of value iteration and Q-Learning, I began to understand the underlying concepts. Then I adjusted my agent to use solely value iteration. After playing with this I moved to Q-Learning and finally deep Q-Learning. There were many roadblocks along the way that I think are the source of my new found conceptualization. Once I was comfortable with Q-Learning I began experimenting with exploration strategies. I got the idea from Shiv to build my own 2D maze environment that I could adjust for different scenarios (in the end I started using a randomly generated scene).

GOALS

1. The major theme of this project was seeing how much I invent before needing to use known algorithms and techniques.
2. The ultimate goal for the Q-Learning agent was testing how exploration functions affect how the bot will traverse a maze of obstacles and find targets. Ideally the bot will efficiently explore most positions then exploit what it has learned.

SPECIFICATIONS

The Q-Learning agent used in the final experiments had evolved from the original value iteration agent. As the environments and my understanding changed the agent did as well. One of the key features of the Q-Learner class was its dynamic state space. The state space is initially empty but grows as states are discovered. This was beneficial when classifying the pose of the agent (an (x,y) pair).

The Environment is essentially designed like the environments from OpenAI Gym. The Agent must be reset each episode and performing actions is done by the step function. The environment also provides a mask for the actions in each state forcing the agent to choose a valid move. Most environments would let the agent try to move but would end up in the same position (a waste of a step in my opinion). In addition to this, episodes would not end until the agent found all targets. This allowed the agent to wander for thousands of steps in a confined space.

The environment is virtually the same for every strategy used, with the exception of minor metadata/reward function tuning. Unless stated otherwise the reward function is 5 * the value of the position - (.001 * t). The discount was always set to 0.9 and alpha ranged between 0.1 and 1e^-6.

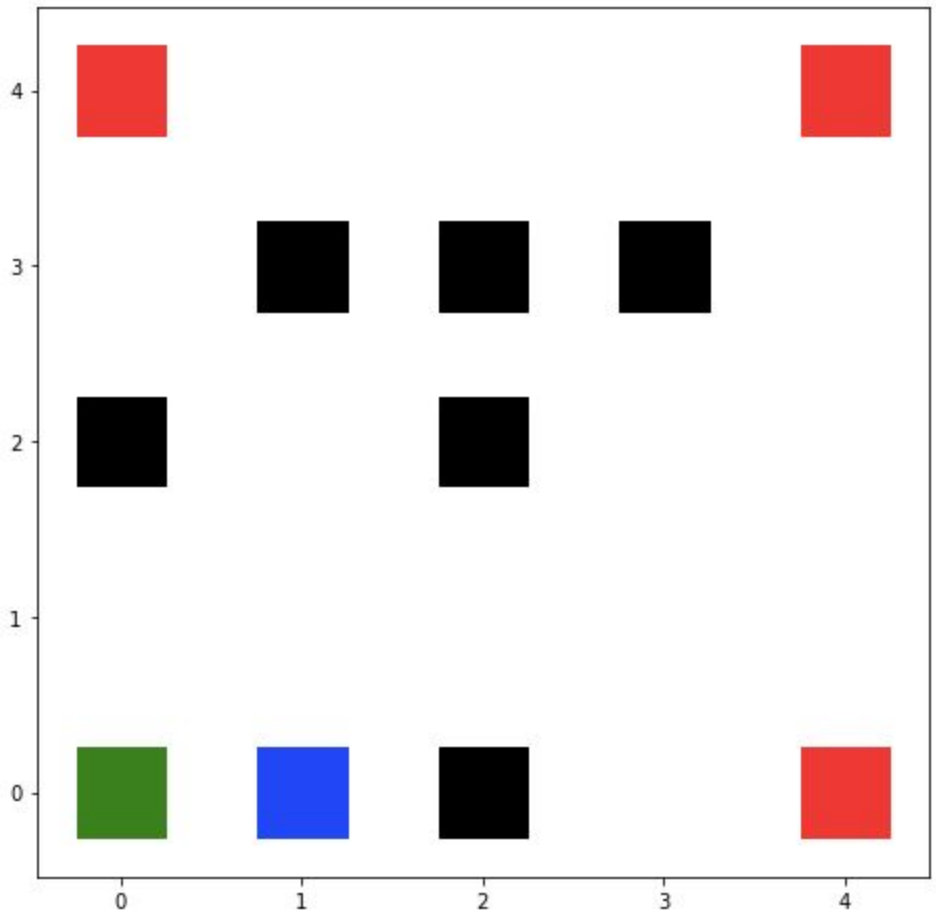

Fig 1. This is a sample scene of a 5x5 maze environment with predetermined obstacles. The goal was to get the agent to find the optimal path to each target. (Blue: agent, Green: spawn, Red: target, Black: obstacle)

Methods

The Runaway

This generation of the agent was only rewarded by its distance from the spawn point. Nearly every iteration this bot would move to the far corner and step back and forth. While this strategy did push the bot away from the spawn it did not help the bot find the targets or avoid obstacles.

The Homeward Bound

This generation of the agent was only rewarded by its distance to the goal. The closer the agent was the higher the reward. As you would expect the agent would move straight to the target. In the scenarios with multiple targets the agent would find an equidistant point and again step back and forth. Like the Runaway this strategy had issues finding targets.

Binary Rewards

This generation was set up to only receive rewards for landing on targets and obstacles. While this seemed like a full proof strategy there was a major flaw. The agent did not get penalized for unproductive exploring. The policy quickly developed cycles the agent would get stuck in and never find the targets.

Epsilon Greedy

This generation used the most common strategy for exploration in RL. The agent randomly performs random actions at a decaying rate. This strategy has a probability of never finding some states, so the agent may miss out on advantages it can't see. Like most of these strategies epsilon-greedy had difficulties dealing with dead ends. This is because, for each position in the alley there is one state that can have one optimal action, either into the dead end or out of the dead end. The dead end scenario was an issue for all the strategies except long memory.

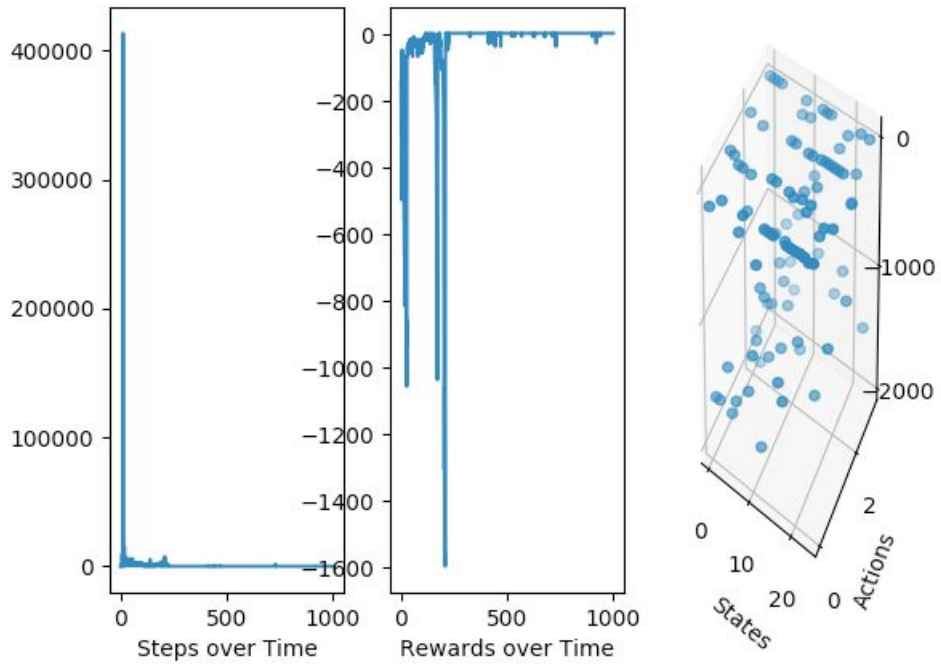

Fig 2. These are the results of running epsilon greedy with epsilon = 0.9 and decay 0.5.

Threshold Based Frequency

In this generation the agent keeps track of the frequency of being in a state and taking an action. The agent would choose an action if its frequency was below a threshold, otherwise the agent would follow its policy.

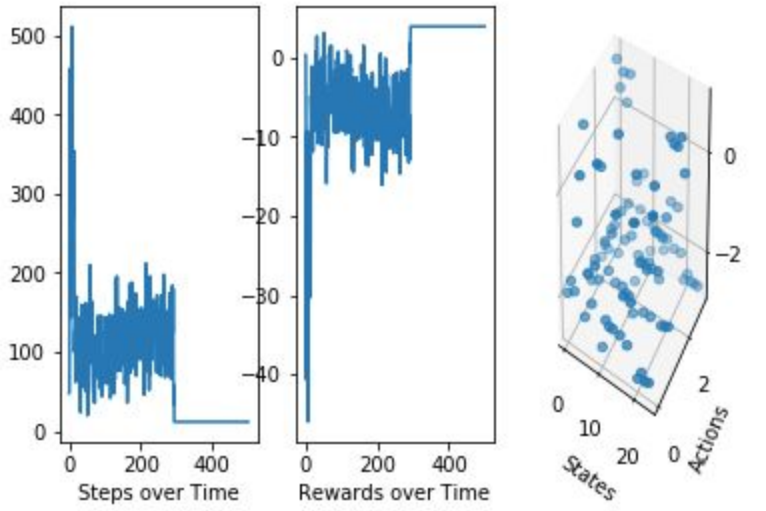

Fig 3. These are the results of running threshold based frequency exploring with a threshold of 50.

Relative Based Frequency

In this generation the agent would choose an action that was below a threshold (# times action taken from this state / # times state has been seen). Once the total frequency of the state was over another threshold the agent would select the optimal action.

Longer Memory

This generation of agents created its state from the previous two poses. Rather than the state just representing its position, it represented its change in position. The inspiration behind this was to help solve the dead end dilemma, the state of entering a dead end is not the same as exiting, which allows for a policy to maneuver this scenario.

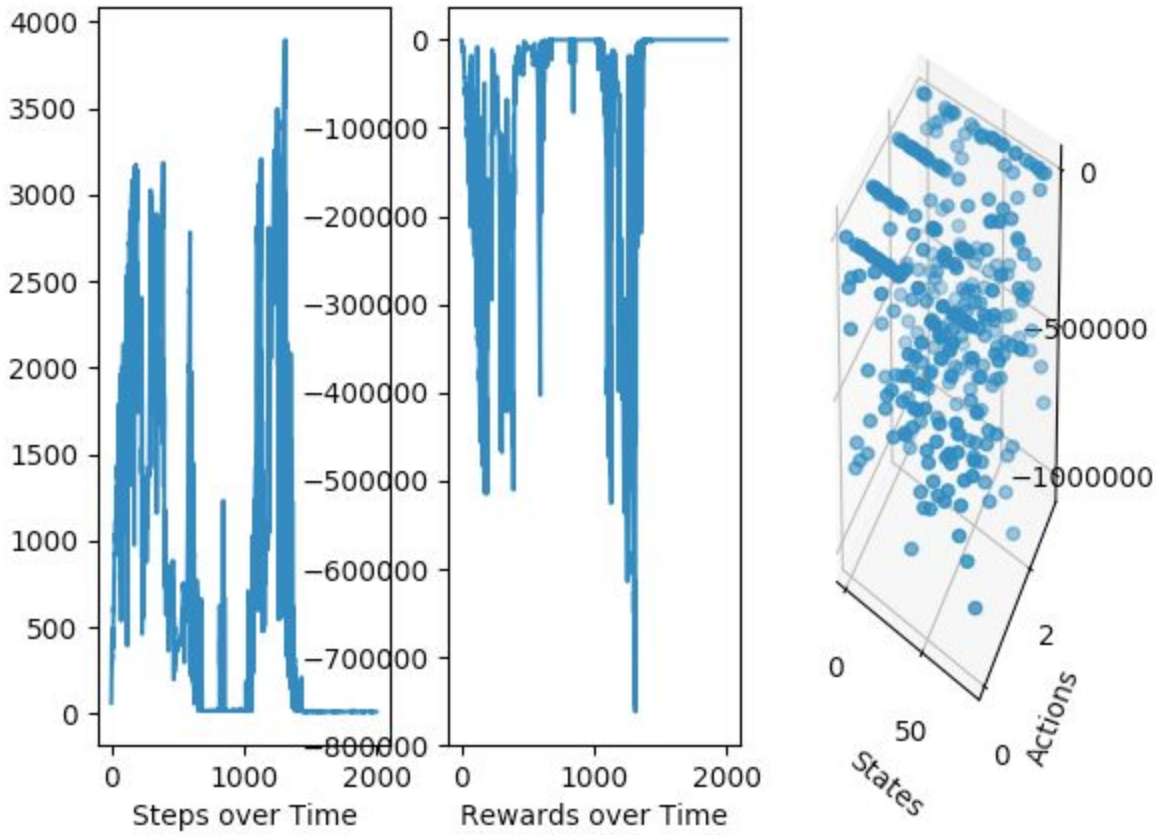

Fig 4. These are the results from running an agent with longer memory and epsilon-greedy

Discussion

The goal of this project was to test the impact of different exploration strategies. There are many supported strategies such as UCB, softmax, pursuit and epsilon-greedy, but the purpose of this project was to gain understanding as to how these work. Using my own recipes for exploration and a simple environment, I hoped to see an agent find the optimal path. While almost every method (with the exception of relative frequency) resulted in the fastest path (12 steps around the perimeter), yet none were able to find the optimal path. At the bottom center of figure 1 there is a lone obstacle against the wall. Ideally the agent would travel around this obstacle and continue its path. However the agent disregarded the total reward and would always travel through the obstacle regardless of the exploration strategy. In addition to this fluke there were times (usually when the time penalty was off) where the agent would get stuck in a loop following the optimal action. I did not find a solution for this problem other than adding the time penalty which seems to work fine. In conclusion the best exploration strategies tested were greedy epsilon and threshold based frequency, these two converged to 12 steps in about 200-400 episodes.

GitHub: https://github.com/MithellScott/Robotics-final-Project