Ben Meares

Professor Palay

COMP560

24 April, 2020

560-Project 2 - Reinforcement Learning

Part 1 - Model-Based Policy Searching

I implemented policy searching. This process had three primary steps: calculate the transition values, get the reward values of each state-action pair, and then calculate the utilities from those reward values. I used a simple 10000 loop per state-action pair iteration to determine how many times each action was determined. I determined the 10000 loop number because the input was rounded to two decimals. To get the reward values, I implemented a recursive function titled “remainingHits” which calculated the number of strokes it usually took to get the ball “In” after taking each action. This exploration function would use the calculated transition values to play the game of golf, choosing actions in each state randomly. Using data gained from this function, I calculated the average amount of hits after each action that it took for the ball to get in the hole. This information H was used as the reward data (1/H). I used this reward data to calculate the utility function, which is just the action with the maximum reward value in each state. This all-encompassing reward value allowed me to calculate the utility function easily.

Part 2 - Model-Free Q-learning

I implemented Model-free Q-learning. I managed a function which took actions to calculate Q-values, which are related to the utility function. These Q values were then used to calculate a utility function. To implement Q-learning, I used a simple explore function that required each state-action pair to be run N amount of times. After this N amount for each action

A in the state S, the state would choose the action with the highest Q value. I stored most variables as class-wide variables, and managed them within the Q-LearningAlg function (mostly). Changing the exploration value to learn towards more exploration caused the program to be more consistent. Adjusting the epsilon value determined whether or not my algorithm randomly chose actions or chose them based on the highest Q-value. The model performed better when the epsilon was lower, and more random actions were chosen. My Q-values seemed to grow exponentially whenever allowed to learn freely (when Alpha > 0.1). The discount value changed my answers to favor the earlier rewards in the process (Fairway to the best action). However, this did not have a noticeable difference in the utility function. I decided to stop learning at an arbitrary number of iterations. I had originally attempted to implement a Checksum system of checking my Q-values to see if they would stop coverging after some N iterations. However, I could not get them to do that given the current constraints and had to abandon the strategy. Note: I only calculated Q-values for each state-action pair, not state-action-state triplet, as that did not make as much sense to me. In-function rewards were used to calculate the benefits of the succeeding states, as outlined in the Q Learning Agent formula/pseudocode in the textbook.

Responsibility: I, Ben Meares, did all of the code since I am the only one in my group (lol)