Bradley Dahlke

CSC 449 – Advanced Topics Artificial Intelligence

Dr. Pyeatt

December 2nd, 2022

Program 3 – Mountain Car via SARSA with Linear Function Approximation

In this program, the objective was to create an artificial intelligence program that could solve the Mountain Car problem using Linear Function Approximation, or SARSA Lambda. The Mountain Car is a classical AI problem that can be described as a car that is stuck between 2 hills that differ, with one of the hills having the goal flag at the top. The objective is to train the artificial intelligence to develop a way of getting the car to the top of that slope and to the goal post, and the way we do that is by saying that every action the agent takes without resulting in a finish is a reward of -1, while the goal post has a reward of 0.

While the code was supposed to be over linear approximation using SARSA lambda, however, this code I assert is more oriented towards the linear approximation of the Q-learning side of things: We are still binning each of the variables in our space to discretize them, then when the agent goes to choose an action it takes from the epsilon greedy way. The way I have written this code is with some help from Phil Tabor, who is an online content creator. Phil's YouTube channel found here: https://www.youtube.com/@MachineLearningwithPhil. While my code was created with a basis on his work, I had some version differences, and could not seem to debug these errors in the proper necessary way as to run the program and get it working like he had his. These errors specifically centered around one mostly, that of which I describe below:

Known Bugs and Errors

Currently, the code does not compile properly. The main issue is with the way I have the binning of the variables, and when the program tries to send the velocity space into the digitize function (in order to discretize the continuous values of this space) it complains that I am sending it a jagged array; Oddly enough, the position bins worked perfectly fine, because those were finished right before where it breaks in the code. This may have something to do with the way specifically the resulting values come out, as like I have mentioned above, the position works, but the velocity does not.

When the code is working, the idea is to get it to bin our state, being the position and velocity, so that it is easier to see where to go from that state. When we don't get this working, however, is when things start to go awry, as stated mainly above. Because of this paradoxical state, there could be numerous other bugs of which the cause is human error that could be easily fixed that will never be found, because at this stage of the program the compiler just stops at this issue. Therefore, it would be impossible to tell if there are any more bugs about, as this is the main focal point stopping us. For preface, the attached picture is in no way what I got out of my plot, but this is something of what it should look like. Attached below is a plot of what the agent Should be producing, where we can see it start to converge to the top of the plot.

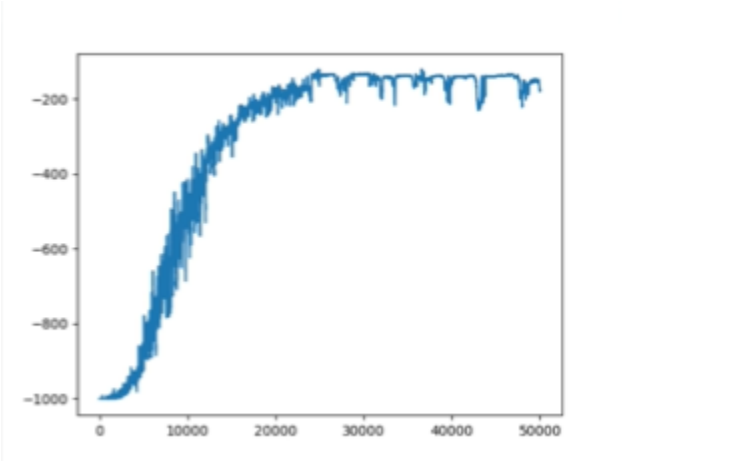

Figure: What the graph should look like

Overall, it was difficult trying to get this to work. Python is definitely not my favorite language ever, and my experience with it has really shown in this semester when it comes to the programming in the later assignments. There is a lot going on for me at this time so hopping in and trying to learn a new language was most definitely one of the most difficult parts. However, understanding the concepts is the main reason I wanted to take this course and I feel that even though I may not be a solid Python programmer, I am sure that I have gotten a way better grasp on these topics thanks to this class.