CSCI 3202 Final Project

Git : https://github.com/steve303/AI_gym.git

Steve Su

My goal was to use an approximate Q learning approach to solve the various games in the open AI classic environment. The environments included the mountain car, both non-continuous and continuous, and the acrobot. Since the states were not discrete I thought approximate learning would be better versus the Q learning method where each state, action pair is stored in a table. Since the states are continuous values this table could get rather large depending on how fine of a resolution one decided to discritize the state values.

Algorithm approach

The first step in approximate learning is to define the Q function which will approxiate the Q value given the state and action. This will be a linear equation in the form: Q = w1*f1(s,a) + w2*f2(s,a) + … wn*fn(s,a) where each weight is multiplied with a feature or basis function, as the textbook calls it. It is helpful to have domain knowledge of the problem when designing the basis functions. We want to maximize the Q value when we are in a good enviroment/state combination. More details will be discussed in the individual environments regarding the design of the basis functions.

Environments

Mountain Car – non continuous

The goal of this game is to drive a car up over a mountain top. The car starts in a valley and must use momentum of both mountain slopes to get over the top. The actions by the agent are push left, no action , and push right and were encoded as 0, 1, and 2, respectively. The state variables given from the environment are position and velocity. I created two basis functions. The Q function was defined as: Q = w1*f1(s,a) + w2*f2(s,a). The weights are w1 and w2 and are updated in the training process. Basis function one was defined as: f1(s,a) = [s(new position) – s(old position)] * (action -1). This equation will give a high score if the car is moving right and the action is push right and vise versa for the left direction. The second basis equation was: f2(s,a) = s(velocity)*(action – 1). In this case, high values are produced when velocity and direction of push are in the same direction. During training of the model the weights will get updated to minimize the loss function (MSE) and hopefully do well enough to drive over the mountain.

In my algorithm, I set up a loop to loop over games to learn the best weights. Note, the learning rate was set at 0.05. I saved the weights of the best performing game and created another function to play it with the weights fixed. There is some randomness as the two games will not behave identically. To give an idea how the algorithm performed, after five games, I had scores of -150, -98, -85, -154, and -157 which are all passing. One must finish before 200 steps, so anything below -200 is passing. Here is a link to the video.

https://www.dropbox.com/s/rsyzrpmzl9dom2t/mtcarV0.mp4?dl=0

Mountain Car - Continuous

This enviroment is very similar except that the actions are continuous. Now the right input is a positive float and left input is negative float. Zero is no input. For this problem I discritized the velocity into 9 bins. They were, -1.0, -0.75, -0.5, -0.25, 0.0, 0.25, 0.50, 0.75, 1.0. For this problem, this seemed to be good enough. Finer resolution should give better results but of course requires more computation. My getAction() function now has to loop through all nine actions compared to three in the non-continuous car game.

The two mountain car enviroments were very similar and I was able to just make minor tweaks to accommodate the continuous environment. The performance scores after five loops were 91.8, 89.1, 92.0, 92.5, and 91.9, which are all passing scores. See link to video.

https://www.dropbox.com/s/gi45nlq3cpm7fy9/mtcarContinuousV0.mp4?dl=0

Acrobot

The last game I experimented with was the acrobot. To design a good basis function took a lot more thought, and I still do not have a optimum design yet. This game environment consists of two freely rotating links. One end of link one is attached to ground and the end of the second link is attached to the free end of link one. The object of the game is to swing the second link above the horizon to win. The agent has three inputs, positive rotation, negative rotation and no action. The environment responds by outputting the angular positions of the links and their velocities as a list of six variables. They are a little confusing and I am a little unsure if I have interpreted the positive and negative directions correctly. Using the same methodology as the first two games, I created several basis functions for the approximate Q equation. My first basis function identified when both links are rotating with the same direction and when link two is positioned in line or ahead of link one. If the rotation is positive, meaning counter clockwise, the action should be positive torque and vise versa for negative rotation for both links. This should increase the basis function with this state/action combination. The second basis function looked at when the links are moving in opposite directions. In this case the action should follow the velocity direction of the second link, meaning apply the same direction torque as the second link. I tried other basis functions but reduced it to these for the time being. I also added a multiplying factor to these basis functions but noticed that sometimes the weights would grow exponentially. I later reduced the learning rate from 0.01 to .0005 to address the exponential growth of my weights. They would climb to inf. Printing out and monitoring the weights and Q values helped to identify some of the problems of the algorithm. Later I was able to pass the game but as you can see it is certainly not optimized.

https://www.dropbox.com/s/dnthao3zl828kfk/acrobot.mp4?dl=0

## Conclusion

I think these games are a great way to test out our AI algorithms. It seems that having some domain knowledge about the enviroment is key to developing good basis functions. I struggled with coming up with decent designs for the acrobot. On the other hand, I could train longer to see if the weights could find a suitable solution for poor basis functions. However, I think its better to work a little bit longer on the basis functions rather than letting the computer run all day. In the end it's a balance of the two.