HW #4 Report

Fateme Noorzad
810198271

For implementing this policy gradient part, RBF part is used. Meaning all the parameters for position and velocity is set same as that part. These values come from the equations state in Sutton book. (page 245)

But here RBF implementation happens for both state and actions.

Because there are 3 actions, sigma of actions is a 3*3 matrix while for states it is a 2*2 one. Another change is that the RBFs are shifted to in the positive part of surface. That's why for position 1.2 and for velocity 0.07 are added. In order to make it a more smooth transition after this shifts both of these values are divided by the total length of the period in which position and velocity are changed.

Because the variation in velocity's value is less than that for position, the number of centers set for it is fewer than position. For position it is 5 and for velocity it's 3. (These values are found by running code multiple times and finding in which way car reaches the top of mountain faster. As an example for velocity instead of 3, 2 was used and it took about 50 more seconds for car to reach the goal.)

Since we what a policy gradient algorithm, there are going to be two weights, each representing weights for policy and state values respectively. For the same reason 2 alphas are defined (although have the same value). These values are found using by running code multiple times and finding in which way car reaches the top of mountain faster.

Same as the RBF implementation in 'Hands on' example, 'Actor-Critic' algorithm is used and as a matter of fact gamma used for updating changes in every iteration.

Again same as 'Hands on' example, reward of agent is changed. Because no matter what it did, environment gave a reward of -1 to it, which doesn't encourage or punish it for a good or bad move, resulting in no learning, that's why when the agent moves in a way(chooses an action) that its destination(state) is a good one (will result in becoming closer to the goal) a big reward of 200 is given.

Because going near to the flag is enough, learning is done when agent becomes close to the flag which is included in code using the 'while' part.

In the end a time is put which tells us how long will it take for that car to reach the top.

(One point, sometimes out of no where it takes a log time for the car to reach the top, if you got bored, just terminate and try again. It will reach there sooner. I promise!)