Implenetaion of Q-Learning and SARSA Algorithms for Mountain Car V0 Problem

Sina Ghanaat
Dept of Mechanical Engineering,
K. N. Toosi University of Technology, Tehran, Iran

***Abstract*—This is the third of three projects for the "AIES" course instructed by Dr. Esmaeil Najafi at the Department of Mechanical Engineering of K. N. Toosi university of technology. In this project, two reinforcement learning algorithms have been implemented in order to solve the mountain car problem. These two algorithms are Q-Learning and SARSA. After implementation and acquiring the resultant graphs, we shall discuss the functionality differences of the two algorithms.**

Keywords—Reinforcement Learning, Q-Learning, SARSA, Mountain Car V0

I. Introduction

Q-Learning and SARSA are both reinforcement learning algorithms which are provide us an optimal policy (in the case of SARSA a semi optimal policy) over the period of finite episodes. Mountain Car problem is a classic reinforcement learning problem which has been introduced in the project outline. The environment of the problem has been provided to us but there were some slight changes to be done in the environment. The Visual Code Studio is the IDE for this project. Unexpectedly, the rendering section of the environment code was not working on my computer. There were problems with the functions of *numpy* module, therefore, with the aid of actual car mountain environment source code, a new rendering function has been defined.

There are two other main codes here. One is the main Q-Learning and SARSA algorithm Code (Q-Learning.py & SARSA.py) and the other is the Q-table visualization code(qt.py) which is been used for visualizing the converged Q-table for the problem.

Since the problem is defined in the continuous space state, a discretization must be don on the environment. This discretization has been defined as a function with changeable discrete segments in the main codes.

for each desirable number of episodes, the Q-tables are saved as *.npy* data and used for Q-table visualization.

II. Step I

In this step, the noise of the environment which is defined as the probability of the agent's action to be random is suppressed and equals to zero. The epsilon greedy strategy is in effect as well as epsilon decaying. The Discrete Segment (DS), Episode (E), Action per Episode (ApE), Learning Rate (LR), Discounted Factor (DF), Initial epsilon are defined in every step and algorithm in future.

A. Q-Learning

In this part, two Q-Learning algorithms have been implemented. The first one contains 30 DS and has an initial epsilon of 100% along with 30000 episodes of training while the other one has less initial epsilon (10%) and 20 DS along with 1500 episodes. The properties of two algorithms are shown in Table 1. The reward graphs for all algorithms contains three types of graphs. The maximum reward graph (yellow) is the maximum reward agent has gotten within 10 consecutive episodes. The minimum reward graph (green) is the minimum reward which the agent received during 10 consecutive episodes. Finally, the average reward graph (blue) is the average of all 10 consecutive episode rewards.

Table 1. Properties of two Q-Learning algorithms

No.	DS	LR	DF	Initial epsilon	E	ApE
1	30	0.1	0.99	100%	30000	1000
2	20	0.1	0.99	10%	1500	1000

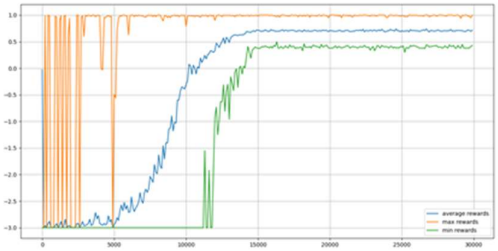

Figure 1. Reward graph for Q-Learning. (DS: 30, LR: 0.1, DF:0.99, initial Epsilon: 100%, E: 30000, ApE: 1000)

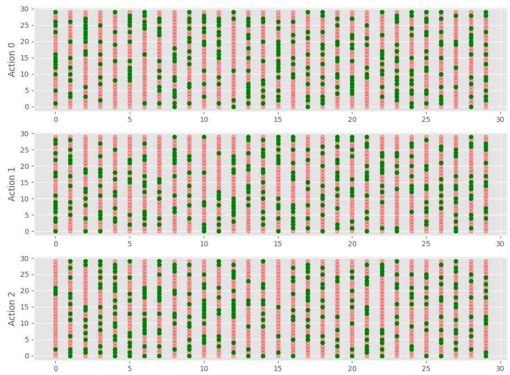

Figure 2. Q-table for the actions at the initial state for Q-Learning. (Y axis: velocity, X axis: position, DS: 30, LR: 0.1, DF:0.99, initial Epsilon: 100%, E: 30000, ApE: 1000)

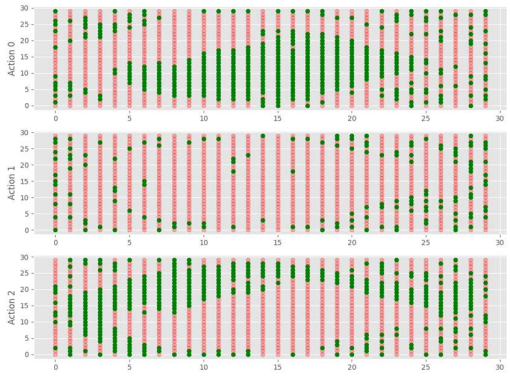

Figure 3. Q-table for the actions at the final state for Q-Learning. (Y axis: velocity, X axis: position, DS: 30, LR: 0.1, DF:0.99, initial Epsilon: 100%, E: 30000, ApE: 1000)

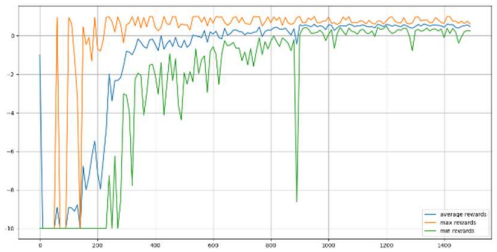

Figure 4. Reward graph for Q-Learning. (DS: 20, LR: 0.1, DF:0.99, initial Epsilon: 10%, E: 1500, ApE: 1000)

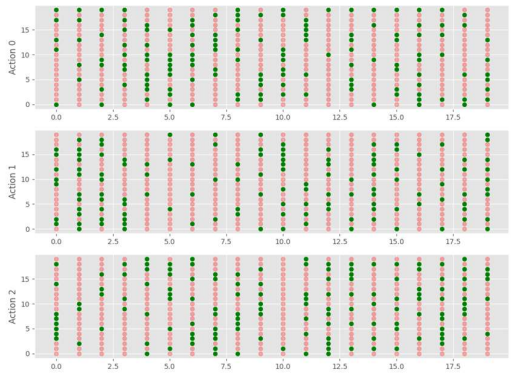

Figure 5. Q-table for the actions at the initial state for Q-Learning. (Y axis: velocity, X axis: position, DS: 20, LR: 0.1, DF:0.99, initial Epsilon: 10%, E: 1500, ApE: 1000)

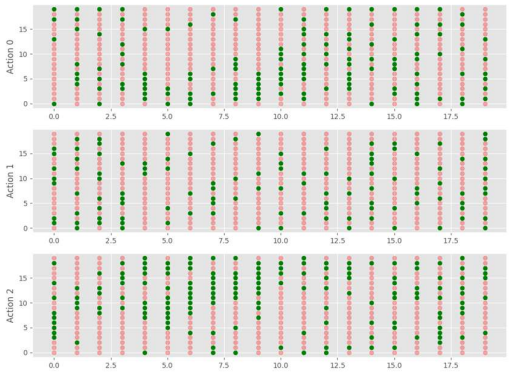

Figure 6. Q-table for the actions at the final state for Q-Learning. (Y axis: velocity, X axis: position, DS: 20, LR: 0.1, DF:0.99, initial Epsilon: 10%, E: 1500, ApE: 1000)

B. SARSA

Same strategy for SARSA except that SARSA will need more exploration while accepting the risk of underfitting. So, the properties for SARSA are shown in Figure 7 caption.

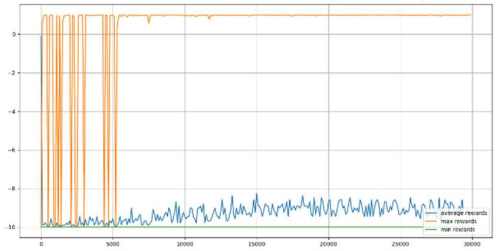

Figure 7. Reward graph for SARSA. (DS: 20, LR: 0.01, DF:0.99, initial Epsilon: 100%, E: 30000, ApE: 2000)

C. Result

The results are interesting. Let's start with Q-Learning. from Figure 1 and Figure 4 we can obviously see that both models have been converged fine. Overall final minimum rewards are near 0.5 for both algorithms which means the agent reaches the flag before 50 steps approximately in each 10 episodes all together. The very small deviation of average reward graph from the other two graphs indicates that agent perform pretty much similar in every initial condition reaching the flag at same step number. Since the initial epsilon value for the first algorithm was 1, therefore the learning process takes long time to converge as the agent tries to explore more rather than exploit which can be seen vividly in Figure 1. But since the initial Q-table is random, any action for the agent is considered as a random action regardless of epsilon value. Therefore, by lowering the epsilon value to 10%, we can reduce the time for learning. Subsequently, by lowering the DS to 20, we can reduce the learning episode further more. This can be seen in second algorithm. An interesting note to make is that in Figure 4, there is a steep decline in minimum graph. By investigating this point, we can see that in that specific episode, the agent was exploring although the epsilon decayed near zero. The Figures 2, 3, 5 and 6 show the initial and final action-based Q-table. The convergence of both algorithms can be seen. Since the far left and right of the map are rarely touched by the agent, those states' action did not change dramatically. another interesting fact is that the action 1 which is do nothing is almost not in action.

However, SARSA algorithm did not converge at all. The Figure 7 illustrated that in all 100 consecutive episodes, there are not a single batch of 100 episodes which in all of them the agent reached the flag. On the other side in every batch of 100 episode there is a sing episode or more which the agent has reached the flag. So, no bullseye for the agent in all its effort. In addition to this, the average reward graph is mostly fluctuating near -9 and bounces near the minimum reward graph. This means that in the most episodes, the agent is not touching the flag and takes too much steps to do so. This is almost related to what SARSA is. The problem is that, although all the individual updates are being done correctly, the following scenario occurs.

The update equation used is the following:

$$Q(s_1, a_1) \leftarrow r_1 + \gamma . Q(s_2, a_2)$$

Now, any update to function approximator means that the Q value changes not just for the updated (state, action) pair, but for all (state, action) pairs.. Now, since our function approximator is altered, next time we use the previous equation for updating any other state, we use the following equation:

$$Q(s_3, a_3) \leftarrow r_3 + \gamma . Q(s_4, a_4)$$

But since Q has itself been changed, the target value of the record 3 changes. This is not desirable. Over time, all the changes cancel each other and Q value remains roughly the same.

III. STEP II

A. Q-Learning

The foundation of this step is as same as the 1st step except for the noise introduction.

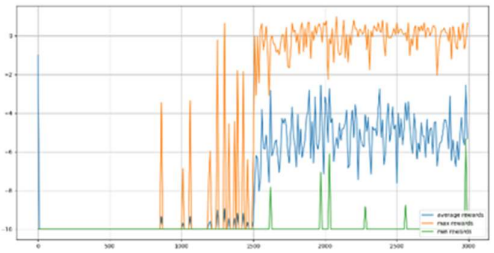

Figure 8 Reward graph for Q-Learning. (DS: 20, LR: 0.05, DF:0.99, initial Epsilon: 10%, Noise: 20%, E: 1500, ApE: 1000)

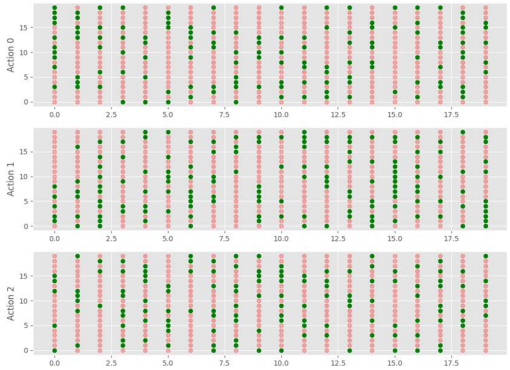

Figure 9 Q-table for the actions at the initial state for Q-Learning. (Y axis: velocity, X axis: position, DS: 20 LR: 0.05, DF:0.99, initial Epsilon: 10%, Noise: 20%, E: 1500, ApE: 1000)

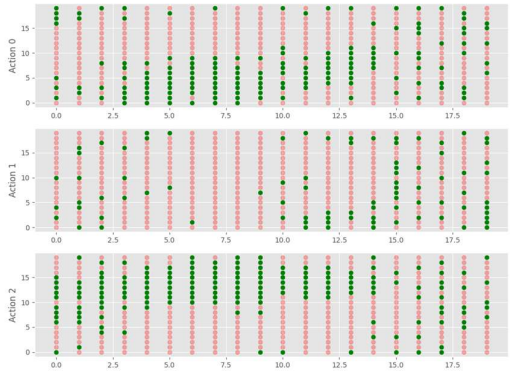

Figure 10 Q-table for the actions at the final state for Q-Learning. (Y axis: velocity, X axis: position, DS: 20, LR: 0.05, DF:0.99, initial Epsilon: 10%, Noise: 20%, E: 1500, ApE: 1000)

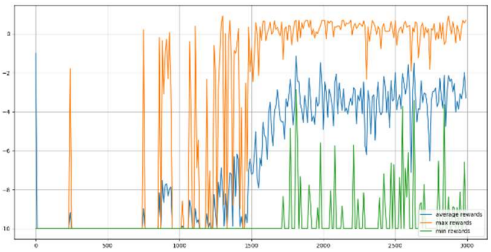

Figure 11 Reward graph for Q-Learning. (DS: 20, LR: 0.05, DF:0.99, initial Epsilon: 10%, Noise: 30%, E: 1500, ApE: 1000)

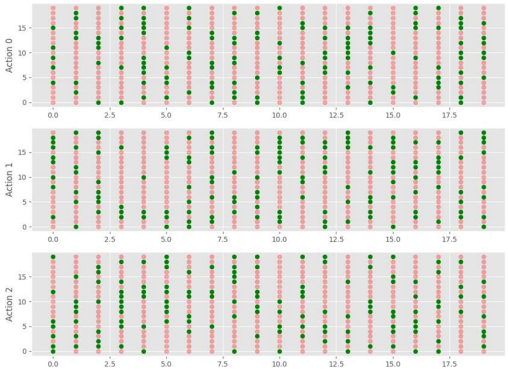

Figure 12 Q-table for the actions at the initial state for Q-Learning. (Y axis: velocity, X axis: position, DS: 20, LR: 0.05, DF:0.99, initial Epsilon: 10%, Noise: 30%, E: 1500, ApE: 1000)

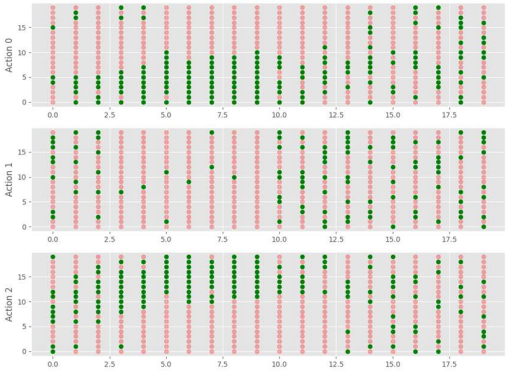

Figure 13 Q-table for the actions at the final state for Q-Learning. (Y axis: velocity, X axis: position, DS: 20, LR: 0.1, DF:0.99, initial Epsilon: 10%, Noise: 30%)

B. SARSA

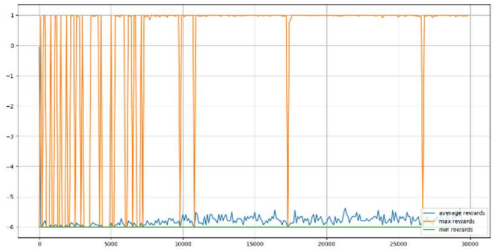

Figure 14 Reward graph for SARSA. (DS: 20, LR: 0.01, DF:0.99, initial Epsilon: 100%, Noise: 20%, E: 30000, ApE: 600)

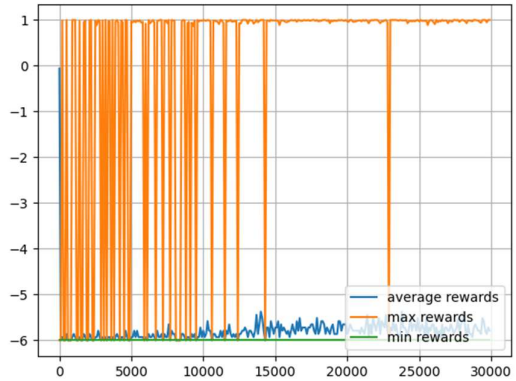

Figure 15 Reward graph for SARSA. (DS: 20, LR: 0.01, DF:0.99, initial Epsilon: 100%, Noise: 30%, E: 30000, ApE: 600)

C. Result

The results are interesting. Let's start with Q-Learning. from Figure 8 and Figure 11 we can obviously see that both models have been converged fine while the 2nd model is not good as the 1st one. Overall final minimum reward for the 20% noise is 0 except for some episodes. On the other hand, for the 30% noise, fluctuation about zero is more sever while minimum reward graph touches zero less than the 20% algorithm. The noise effect on the algorithm can be seen on the inevitable fluctuation in the reward graphs The Figures 9, 10, 12 and 14 show the initial and final action-based Q-table. The convergence of both algorithms can be seen. The 2nd algorithm clearly shows less convergence than the 1st one and randomness can be seen in the actions. One of the noise effects on the model is that the far left and right of the map which were rarely touched by the agent, now get touched more and because of the random noise actions.

However, SARSA suffers from the same issue as mentioned above.

IV. STEP III

The aforementioned two steps were implemented with the best learning rate and discount factor; therefore, this step has already been done.