Mountain Car 2D

Abstract

This work propose to investigate the Reinforcement Learning algorithm Q-Learning in the Mountain Car Problem. Reinforcement Learning (RL) is a successful technique for learning the solutions of control problems from an agent's interaction, the policy is learned through trial-and-error interactions of the agent with the environment. Reinforcement Learning is a paradigm that is very successful when used by agents trying to maximize some notion of cumulative reward, as they carry out sensing, decision, and action in an unknown environment.

1 Mountain Car 2D

The Mountain Car Problem [1] is a domain that has been traditionally used by researchers to test new reinforcement learning algorithms. In the Mountain Car Problem a car that is located at the bottom of a valley, where the goal is to get the car to drive out of the valley. To the drive out of the valley the car must be pushed back and forward until it reaches the top of a hill. The agent must generalize across continuous state variables in order to learn how to drive the car up to the goal state.

The first time that mountain car problem show was in the Andrew Moore's PhD Thesis in 1990. In 1998 when Sutton and Barto added the mountain car problem in his book, the problem became more widely studied and year after year the problem received different versions, with new kinds of reward functions, start/end states, etc.

In 2D mountain car problem (Figure 1) two continuous variables describe the agents state: the horizontal position (x) restricted to the ranges [-1.2, 0.6] and velocity ($\dot{x}$) restricted to the ranges [-0.07, 0.07]. The agent may select one of three actions on every step: Left, Neutral, Right, which change the velocity by -0.0007, 0, and 0.0007 respectively. The update function is $velocity = velocity + (Action) * 0.001 + cos(3 * position) * gravity$ and $Position = position + velocity$.

One important aspect in my opinion about the 2D Mountain Car learning is about the agent(car). The car needs to learn to go away from the objective to take impulse to go out of the valley, this is only one of few problems that need this kind of action.

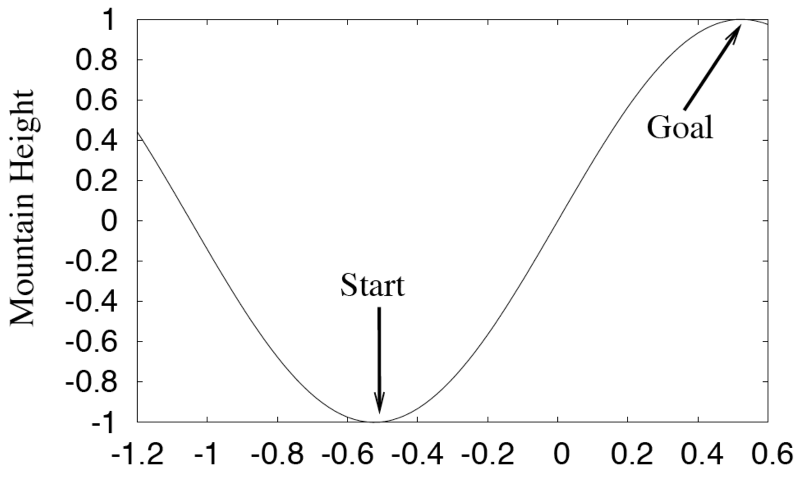


Figure 1: The 2D Mountain Car Problem. (Image from [2]).

2 Reinforcement Learning, Q–Learning and S.A.R.S.A. algorithm

Reinforcement Learning (RL) algorithms have been applied successfully to the on-line learning of optimal control policies in Markov Decision Processes (MDPs). In RL, this policy is learned through trial-and-error interactions of the agent with its environment: on each interaction step the agent senses the current state s of the environment, chooses an action a to perform, executes this action, altering the state s of the environment, and receives a scalar reinforcement signal r (a reward or penalty).

The goal of the agent in a RL problem is to learn an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maps the current state s into the most desirable action a to be performed in s. One strategy to learn the optimal policy π^* is to allow the agent to learn the evaluation function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. Each action value $Q(s, a)$ represents the expected cost incurred by the agent when taking action a at state s and following an optimal policy thereafter.

The Q–learning algorithm Watkins [3] is a well-know RL technique that uses a strategy to learn an optimal policy π^* via learning of the action values. It iteratively approximates Q, provided the system can be modeled as an MDP, the reinforcement function is bounded, and actions are chosen so that every state-action pair is visited an infinite number of times. The Q–learning update rule is:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right], \tag{1}$$

where s is the current state; a is the action performed in s; r is the reward received; s' is the new state; γ is the discount factor ($0 \leq \gamma < 1$); and α is the learning rate. To select an action to be executed, the Q–learning algorithm usually considers an $\epsilon - Greedy$ strategy:

$$\pi(s) = \begin{cases} \arg\max_a \hat{Q}(s,a) & \text{if } q \leq p, \\ a_{random} & \text{otherwise} \end{cases} \tag{2}$$

where:

- q is a random value uniformly distributed over $[0,1]$ and p ($0 \leq p \leq 1$) is a parameter that defines the exploration/exploitation trade-off: the larger p, the smaller is the probability of executing a random exploratory action.

- a_{random} is an action randomly chosen among those available in state s.

The S.A.R.S.A algorithm was proposed by Rummery and Niranjan [4] as another way to learn an optimal policy π^*. The main difference between the two algorithms is that the S.A.R.S.A is a on-policy control algorithm, while the Q–Learning is an off-line method. The S.A.R.S.A update rule is:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \hat{Q}(s',a') - \hat{Q}(s,a) \right], \tag{3}$$

This name simply reflects the fact that the main function for updating the Q-value depends on the current state of the agent S, the action the agent chooses A, the reward R the agent gets for choosing this action, the state S' that the agent will now be in after taking that action, and finally the next action A' the agent will choose in its new state. Taking every letter in the quintuple (s, a, r, s', a') yields the word S.A.R.S.A [5]

In RL, learning is carried out online, through trial-and-error interactions of the agent with the environment. Unfortunately, convergence of any RL algorithm may only be achieved after extensive exploration of the state-action space.

3 Experiments and Results

In this experiment/simulation I made the 2D Mountain Car problem with Q-Learning and S.A.R.S.A, for every time step in this simulation the reward was -1 and when the car reaches the top of a hill the reward was $+1$. Explanation about how the Mountain Car works was discussed in 1 and Q-Learning and S.A.R.S.A. was discussed in 2, in the learning algorithm is usual to use the tabular form, but in the mountain car domain is not a good choice to build the tabular form because this domain have to much state and will consume a lot of memory and the learning will be deficient.

To help with this problem I made the use an simple grid localization. In the GRID approach a grid with size of 100 units was built, this grid was used to storage the information, very similar as tabular form and with less memory.

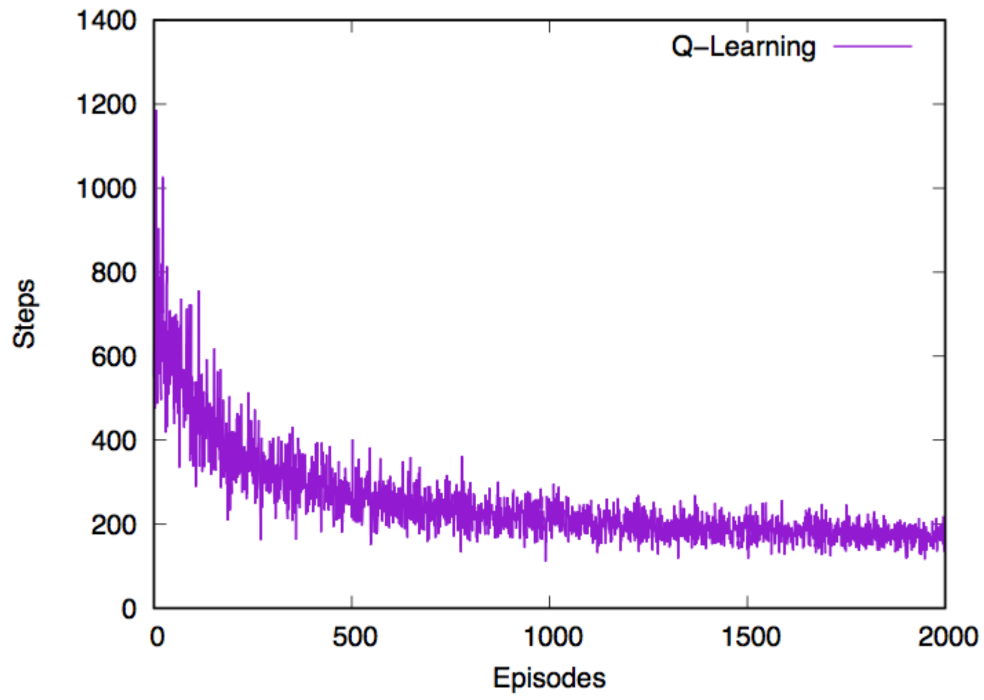
The algorithm received the information by the function *QL(position in grid, velocity in grid, action)* and every step that need to update the Q-Learning, its necessary to "transform" the action value (Q) in an information to be understand by the GRID and vice-versa. The equation is demonstrated by the the equation 4 and 5

$$PositonGRID = \frac{PositionValue - (-1.2)}{0.60 - (-1.2)} * 100 \tag{4}$$

$$VelocityGRID = \frac{VelocityValue - (0.07)}{0.07 - (-0.07)} * 100 \tag{5}$$

The results presented was based in the average of 30 training sessions for each algorithm. Each episode is composed of the number of steps when the car reaches the top of the Hill, 2.000 episodes was made. The parameters used in the experiments were the same for the two algorithms, Q–learning and S.A.R.S.A: the learning rate is α = 0.2, the exploration/ exploitation rate is 10% and the discount factor γ = 0.9. Values in the Q table were randomly initiated, with $0 \leq Q(s, a) \leq 1$. The experiments were programmed in Python 2.7 and executed in a IMAC , with 16GB of RAM on a MAC-OS platform.

Figure 2 - value function table - show the learning curve of the Q-Learning and S.A.R.S.A algorithm. An first comparison is possible to see that SARSA uses less steps in the early of the learning to reaches the objective, but the Q-Learning needs 29 minutes to learning and the S.A.R.S.A needs 35 minutes to do the same job.



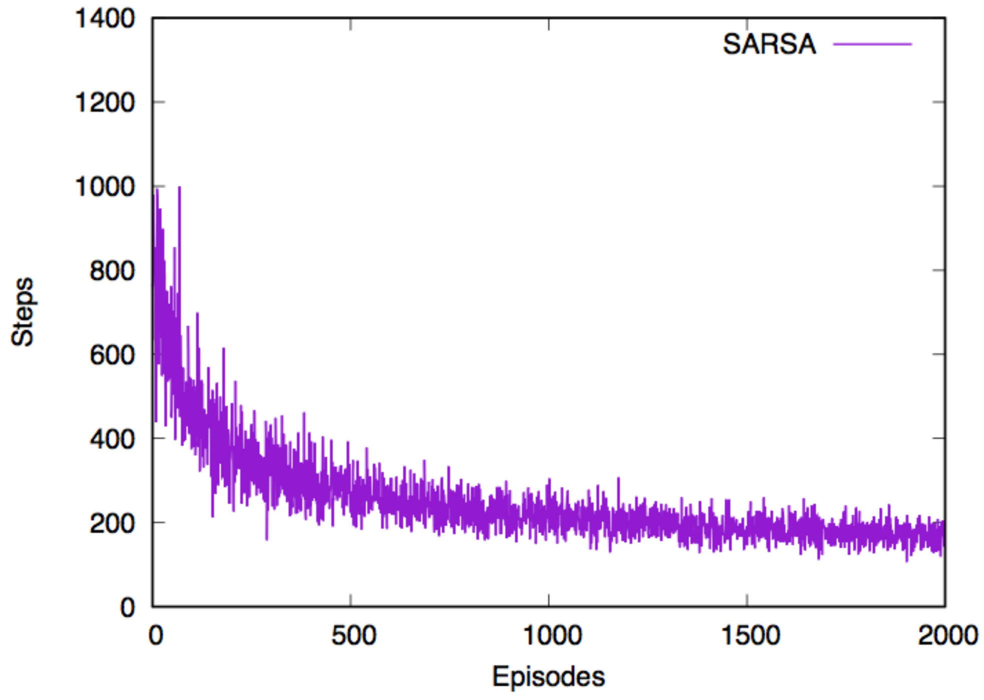


Figure 2: Number of the steps to reaches the objetive using the Q-Learning and S.A.R.S.A algorithm

Student's t–test [6] was used to verify the hypothesis that the use of Q–learning and S.A.R.S.A have or have not differences of the speed of the learning process. The value of the module of T was computed for each episode using the same data presented in figure 2. The results in the figure 3 show that the both algorithm have the same speed (need to be a T value more than 2.45 to be statistical difference)

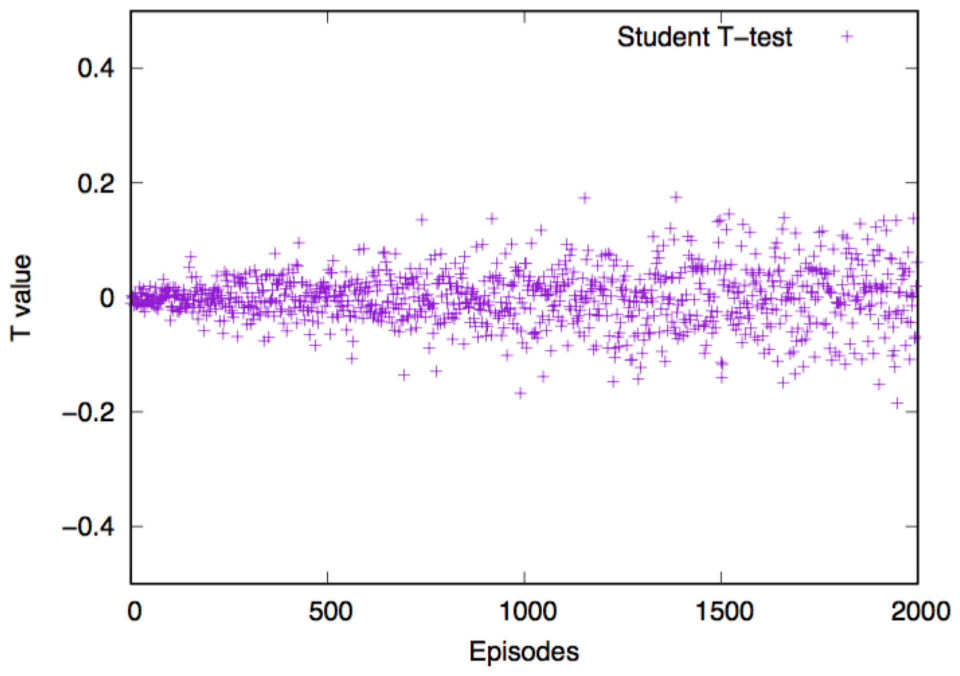
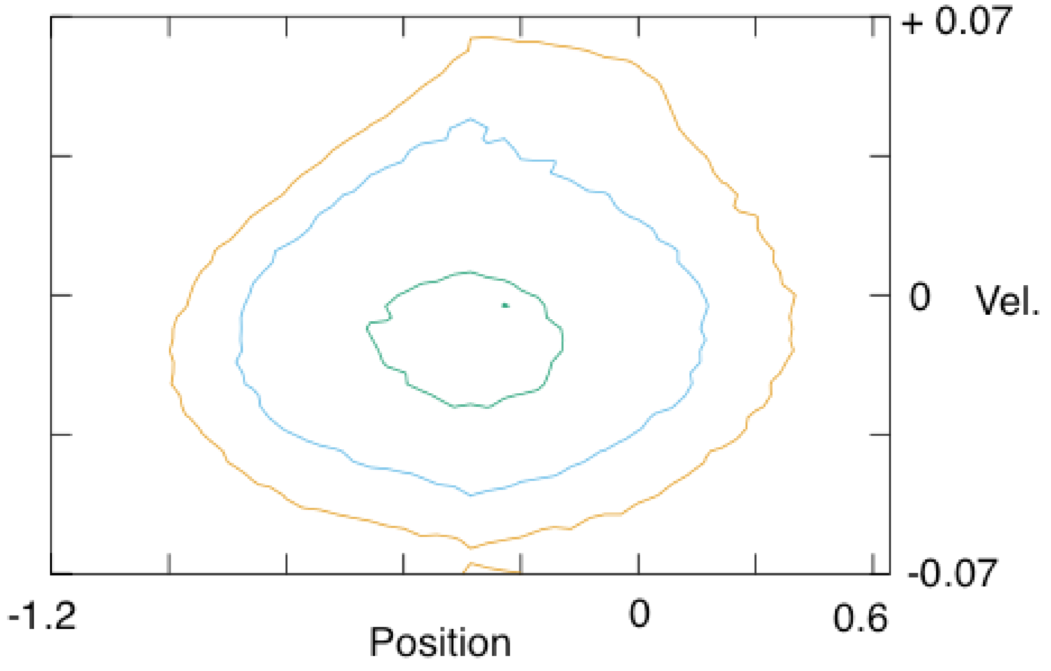


Figure 3: Results from Student's t test between Q–learning and S.A.R.S.A algorithms,

The figure 4 show one important information about the 2D mountain Car Domain. This dominion have the strong characteristic to be a non linear, without existence of the straight obstacles, this characteristic generate a value function where borders are curved. The figure show the Value function on 2D and 3D of the Q–learning algorithms, S.A.R.S.A show very similar aspect.



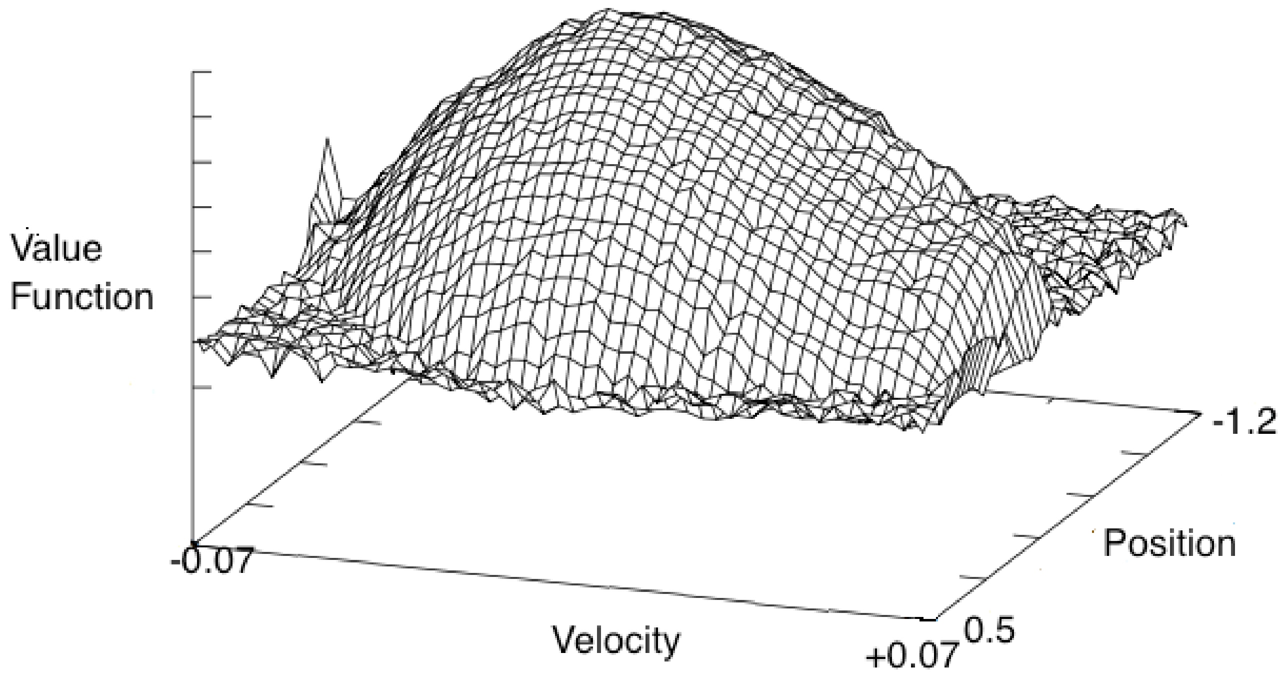


Figure 4: The Value Function table of the 2D Mountain Car

4 Conclusion

This work propose the use of the Mountain car 2D problem with Q-Learning and S.A.R.S.A. The reason to built this domain was because the only domain I found was the C version[1] of 1996 and I decided to built my version of 2D Mountain Car in Python and do tests with Q-Learning and S.A.R.S.A algorithm.

The result was very nice, both algorithm had the same performance, and it was confirmed by the Student's t–test, but it is easy to see that S.A.R.S.A needs less steps to reach the objective in the start of learning than Q-Learning, probably the difference of the steps between these algorithms was because S.A.R.S.A learns the policy and sometimes takes an optimal actions and explores other actions, while Q-Learning takes optimal (estimated) action and learn about the policy that does not explore. S.A.R.S.A will learn to be careful in an environment where exploration is costly, Q-learning will not. These differences made the one algorithm have a slight better performance.

5 Machine Learning Capstone Project: readme

The main directory 2D Mountain Car contain:

- File ***mcar2D.py*** with the problem and algorithms in python;
- Data Directory with QL, SARSA collected results;
- ttest directory inside the data directory with ***ttest.py*** with Student t-test implementation and data;
- Report in PDF with the implementation, discussed algorithm and conclusions about the problem that I choice.

[1]after the work I found some in Python

A Udacity Reviews

A.1 Metrics used to measure performance of a model or result are clearly defined. Metrics are justified based on the characteristics of the problem.

In this section, you will need to clearly define the metrics or calculations you will use to measure performance of a model or result in your project. These calculations and metrics should be justified based on the characteristics of the problem and problem domain. Thus what is the metric/metrics used to measure performance of the agent? Reaching the top? Rewards? Penalties? Etc.. Note: Is the update function your metric? Thus please thoroughly discuss.

Answer: The metric to determine the performance of the Mountain car 2D was the ”steps”. In the graphics is possible to see the number of steps versus episodes. In the code I gave the name of ”iterations”, the number of iterations is the number of steps. Every episode was made of the number of steps necessary to car reach the top of hill and the episode end when the car reaches the objective. Every new episode the car start in the bottom of the valley and need less steps to reach the objective, because he learning how to do that.

A.2 Student clearly defines a benchmark result or threshold for comparing performances of solutions obtained. In this section, you will need to discuss the algorithms and techniques you intend to use for solving the problem. You should justify the use of each one based on the characteristics of the problem and the problem domain.

As what could be a good benchmark here? Are there any online papers or resources that have demonstrated a good benchmark for the Mountain Car Problem? What would be a good update function?

Answer:A good benchmark here is when the car needs a low number of steps to reach the top of hill. For example in the work of Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro and Anna H. R. Costa (Heuristically Accelerated Reinforcement Learning: Theoretical and Experimental Results) [7], they use a Heuristically Accelerated Reinforcement Learning to speed-up the learning and the heuristically "help" the car to reach the top of hill faster. The result of their can see in figure 5. The black line was the Q-Learning, can not see the full learning of Q-Learning but look like similar to the figure 2 of my Q-Learning. The others learnings, was the algorithms it some Heuristically and the faster was the $HA - Q(\lambda)$ with less than 200 steps to reach the objective.

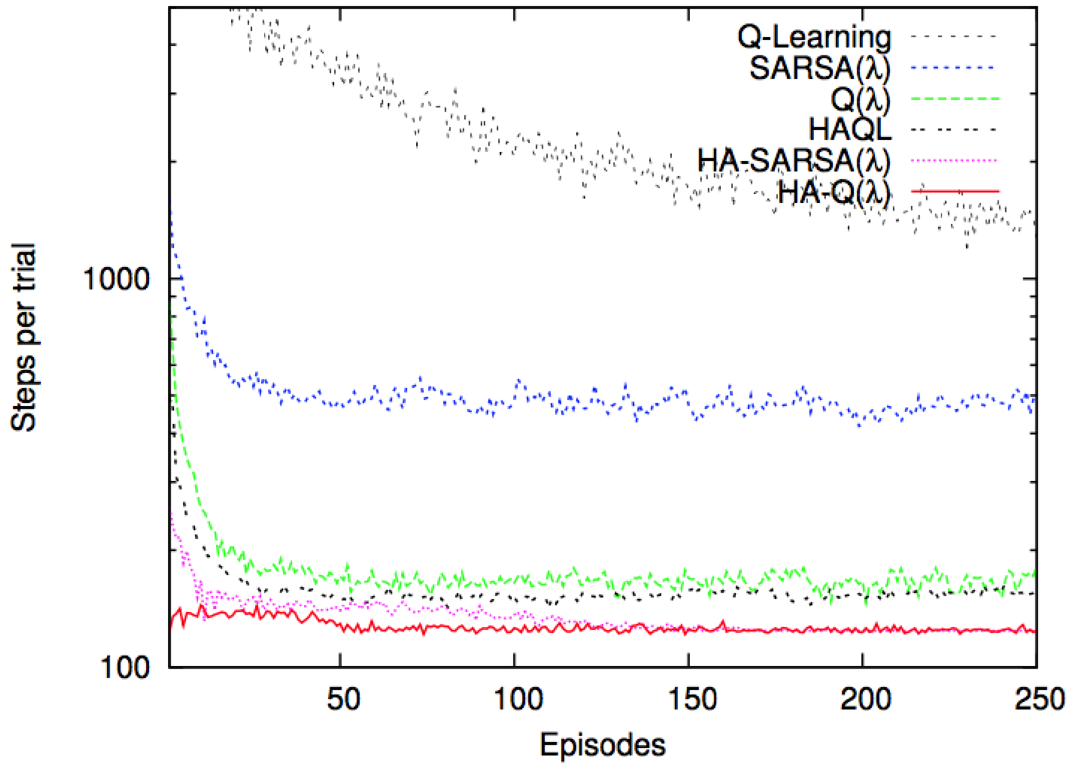


Figure 5: Results of the paper of Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro and Anna H. R. Costa

A.3 All preprocessing steps have been clearly documented. Abnormalities or characteristics about the data or input that needed to be addressed have been corrected. If no data preprocessing is necessary, it has been clearly justified.

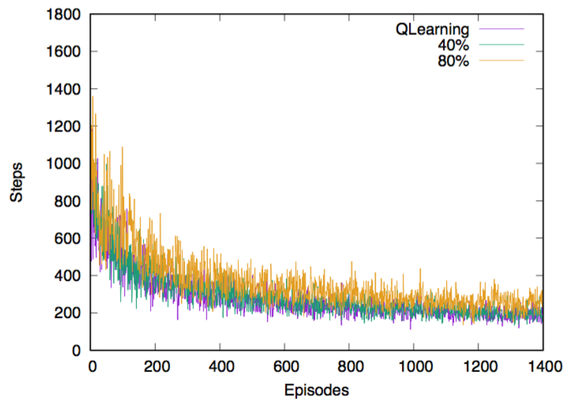
It seems as there were no preprocessing steps done(which is typical with RL problems), therefore please also mention this in the analysis.

Answer:There is no need a preprocessing step here. The car start at the bottom of a valley, after taht, the car choice a action, make the action, receive a reward from action taken and update the Q-Learning. The car choice a lot of actions this process carry on until the car find the objective and receive the positive reward and finally the episodes. Another episode start everything again happens (updating Q-learning) and every time the car start a new episode he is more ”smart” until he don’t need much steps to reache the top of hill. When the number of steps in different episodes was the same it say - the learning was stable. One case is needed a preprocessing if try to do, for example a Heuristically Learning and by the concept of Heuristically it is possible to preprocessing some information to speed up the learning, but I don’t use it in my Mountain Car.

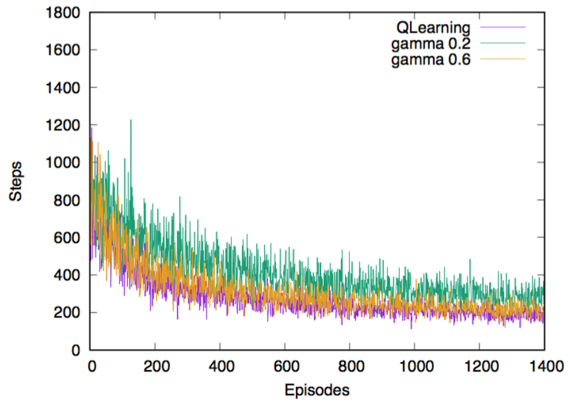
A.4 The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary.

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Therefore how did you ultimately decide to use α *= 0.2, the exploration/ exploitation rate is* 10% *and the discount factor* γ *= 0.9. Values in the Q table were randomly initiated, with* $0 \leq Q(s, a) \leq 1$*.”I would suggest that you could test and tune these values to test how well* *the agent performs. One way to systematically test and verify your experimental findings would be to provide a chart of all the parameter values that were tested and report the numerical performance. And once you find the q-learning agent with the best combination, you can than justify why it might be the best optimal model.*

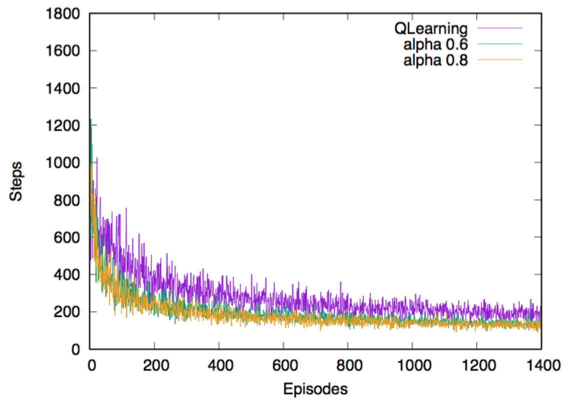
Answer: One brief explanation about how these values change the learning is here [8] with the work of Sutton and Barto. The table was randomly initiated, with $0 \leq Q(s, a) \leq 1$ to help the car explore more easily the environment, these values "force" the car to do randomly action and to start to learning through the reward feedback.



((a)) exploration/ exploitation



((b)) γ values



((c)) alpha values

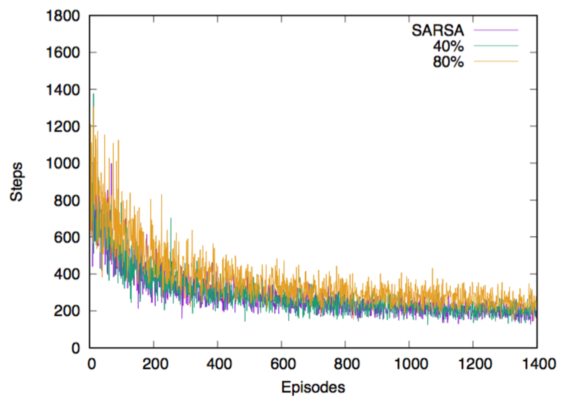
Figure 6: QLearning with default and different parameter values

In the figure 6 the QLearning always have the same configuration (α = 0.2, the exploration/ exploitation 10% and γ = 0.9). The first test was em ”a” with different exploration/exploitation, the results was as expected, because when have a high value of exploration/exploitation more randomness is the algorithm and worse is the performance.

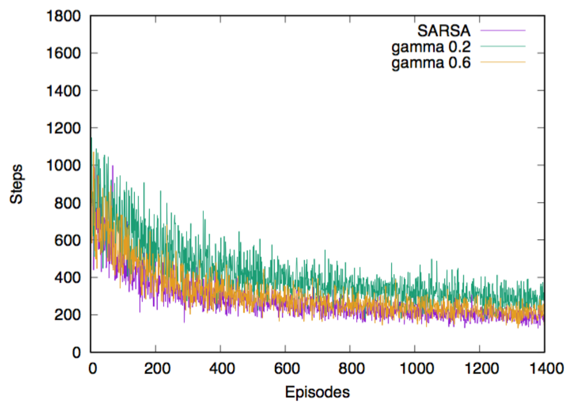
The ”b” the γ was changed. The value of $\gamma = 0.9$ was the best value in this situation. When the gamma is close to “0” the agent tends to consider only immediate values of reward. If the values are close to 1, the agent considers the future rewards with the most weight.

In the last one, the α was changed and showed a interesting result. The ”best” value of α in this kind of problem is 0.8. I believe this height value of α help the car to be more “open” to the information in the early learning stages when the car know nothing and improve the learning.

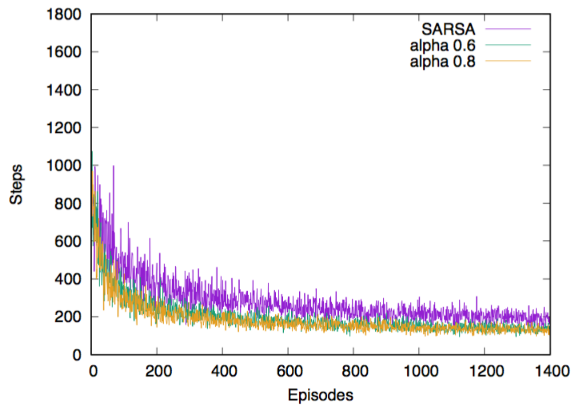
The same experiment was made with S.A.R.S.A, in figure 7 and the results was very similar.



((a)) exploration/ exploitation



((b)) γ values



((c)) alpha values

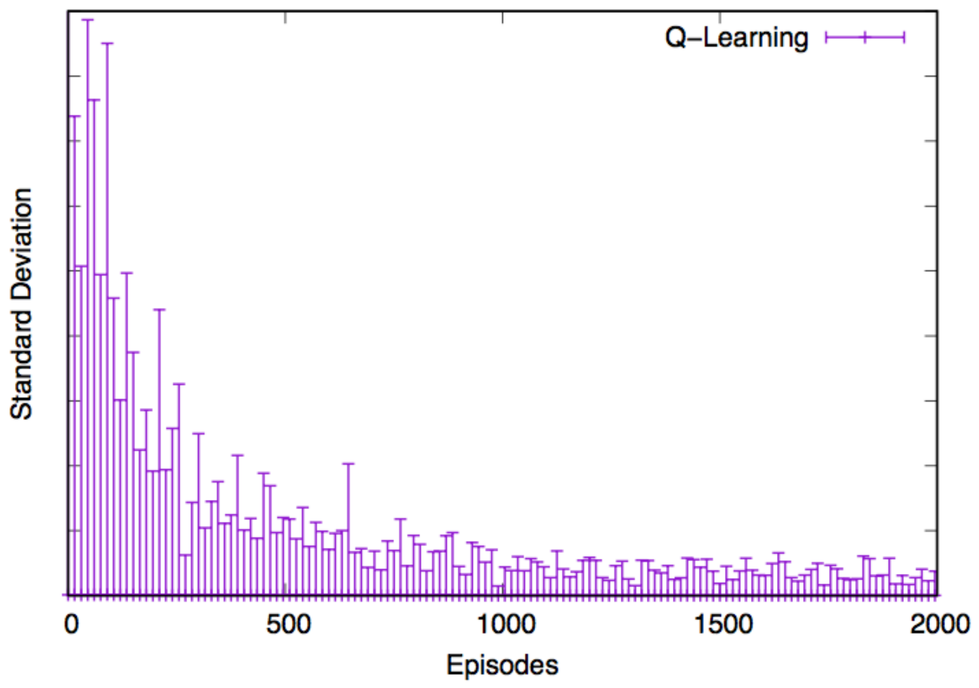
Figure 7: S.A.R.S.A. with default and different parameter values

A.5 The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem.

Once you establish a benchmark, in this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project.

Answer:One statistical analysis was made with Student's t–test to verify the hypothesis about the use of Q–learning and S.A.R.S.A have or not differences of the speed of the learning process. The result in Figure 3 answer with big **NO**, result means the both results was statistical the same, the only thing I mention was about the little time in minutes of both learning's (Q-Learning needs 29 minutes to learning and the S.A.R.S.A needs 35 minutes) but 6 minutes is not enough to affirm anyting about learning speed.

Another, more simples, but important test was the standard deviation of the average of the learning - results presented was based in the average of 30 training sessions for each algorithm. I don't show this results previously but now I show for discussion in figure 8



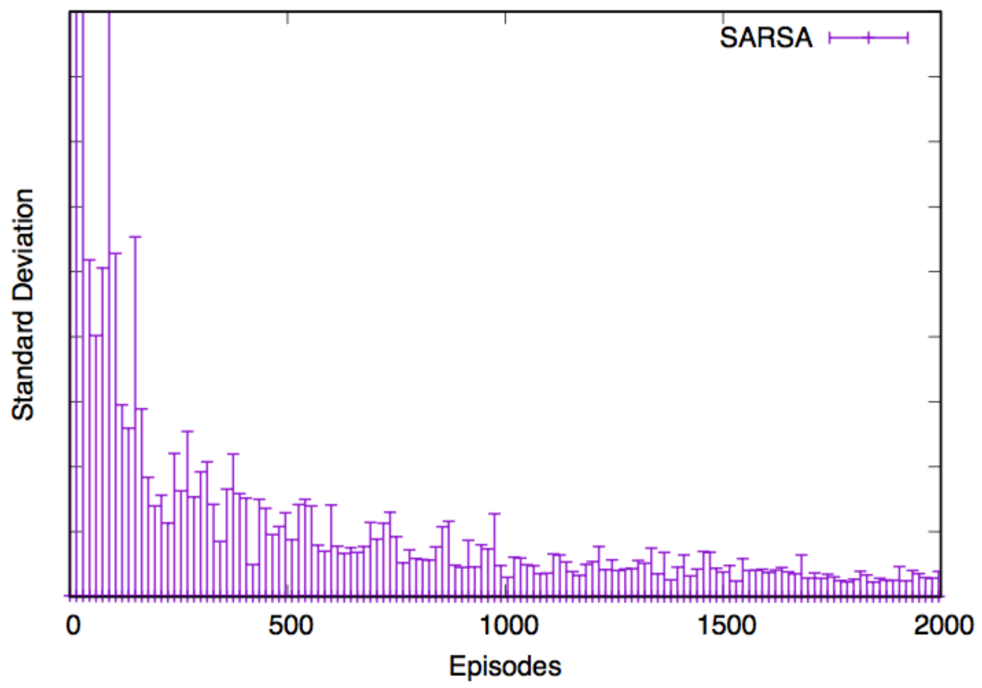


Figure 8: Standard Deviation of 30 training sessions

This Standard Deviation walk through the learning show how the error was became small through the learning, this happened because of variation of the results in the end of learning became very similar (very similar number of steps every time).

A.6 Student adequately summarizes the end-to-end problem solution and discusses one or two particular aspects of the project they found interesting or difficult.

Great job describing your end-to-end problem solution and it is very cool that you "decided to built my version of 2D Mountain Car in Python and do tests with Q-Learning and S.A.R.S.A algorithm." However for this section can you also discusses one or two particular aspects of the project that you have found interesting or difficult? As any issues doing this in python?

Answer: I was interested about the problem, first because in this kind of problem the agent need to learn to go in different direction (to take impulse) to reach the objective, and in my opinion it is a very nice problem to solve. Second, don't have much (I find only one) car problems with Python. Have a lot only the C/C++ problem to download with more then 20 years old. I don't have problem to find theoretical information about the mountain car and the python help a lot to build the system. I took more or less three weeks to build all the functions and one week to solve the bugs, the only thing took some time to build was the value function data to plot, but after I was able to synchronize all the data, the graphic works like a charm.

A.7 Discussion is made as to how one aspect of the implementation could be improved. Potential solutions resulting from these improvements are considered and compared/contrasted to the current solution.

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, *consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution.*

Answer: I am afraid to repeat my self, but the only thing I could do to improve my implementation is to put some heuristics to make the learning faster and to car needs less number of steps to reach the objective. Using the idea of Reinaldo A. C. Bianchi [7] it is possible to improved action selections based on heuristics. The heuristic function indicates that an action must be taken instead of another. To do that this heuristic function is used only in the action choice rule.

The action choice rule used is a modification of the standard $\epsilon-Greedy$ rule used in Q–learning, but with the heuristic function included:

$$\pi(s) = \begin{cases} \arg\max_a[\hat{Q}(s,a) + H(s,a)] & \text{if } q \leq p, \\ a_{random} & \text{otherwise} \end{cases} \tag{6}$$

where:

- H is the heuristic function, which influences the action choice.
- q is a random value uniformly distributed over $[0,1]$ and p ($0 \leq p \leq 1$) is a parameter that defines the exploration/exploitation trade-off: the larger p, the smaller is the probability of executing a random exploratory action.
- a_{random} is an action randomly chosen among those available in state s.

The H is a data preprocessing of learning, for example it possible to build a H-table, similar of Q-table with all information about how to solve one problem, and in every step the agent will be affected by the H value. In his work Bianchi proves that a wrong H don't invalidates the learning, just make less faster, its happen because if the H is wrong the agent will get a negative reward and this action over time will be less important. More about this information can be found in [7] and [9].

References

[1] A. Moore, "Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces," in *Machine Learning: Proceedings of the Eighth International Conference*, L. Birnbaum and G. Collins, Eds. 340 Pine Street, 6th Fl., San Francisco, CA 94104: Morgan Kaufmann, June 1991.

[2] M. E. Taylor, "Autonomous inter-task transfer in reinforcement learning domains," Ph.D. dissertation, University of Texas at Austin, Austin, TX, USA, 2008.

[3] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[4] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994, technical Report CUED/F-INFENG/TR 166. Cambridge University, Engineering Department.

[5] Wikipedia, "State-action-reward-state-action Wikipedia, the free encyclopedia," 2016, [Online; accessed 10-July-2016]. [Online]. Available: https://en.wikipedia.org/wiki/State-Action-Reward-State-Action

[6] M. R. Spiegel, *Statistics*. McGraw-Hill, 1998.

[7] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Heuristically accelerated reinforcement learning: Theoretical and experimental results." in *ECAI*, ser. Frontiers in Artificial Intelligence and Applications, L. D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, Eds., vol. 242. IOS Press, 2012, pp. 169–174. [Online]. Available: http://dblp.uni-trier.de/db/conf/ecai/ecai2012.htmlBianchiRC12

[8] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[9] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Heuristically accelerated q-learning: a new approach to speed up reinforcement learning," *Lecture Notes in Artificial Intelligence*, vol. 3171, pp. 245–254, 2004.