Advanced Concepts in Machine Learning

Assignment 4

Reinforcement Learning

Author:
Tom Dooney (6230256)

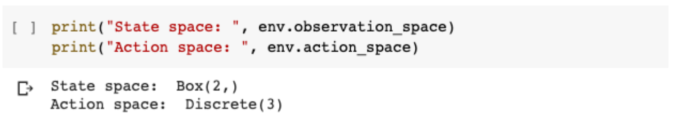
November 26, 2019

Abstract

In this assignment a reinforcement learning (**RL**) algorithm was adapted from an online resource and implemented by means of Q-learning. This algorithm was applied to the 'Mountain Car' problem. The 'Open AI Gym' package [1] was used to generate the 'Mountain Car' environment. A range of parameters were tested to obtain a setup for the algorithm to learn fastest, and Q-tables were plotted to visualise the optimal state-action values to achieve the objective of 'Mountain Car'.

1 README

This implementation of reinforcement learning was adapted from an implementation found on GitHub [2]. This package includes a selection of **RL** environments, one of which being 'Mountain Car'. After this his environment is first generated, the state and action spaces are printed.

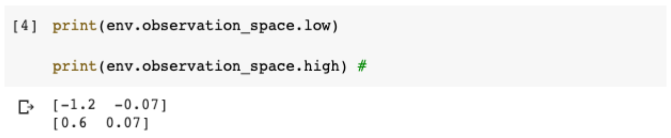


```
[ ] print("State space: ", env.observation_space)
    print("Action space: ", env.action_space)

State space:  Box(2,)
Action space:  Discrete(3)
```

Figure 1: State Space and Action Space Dimensions

Figure 1 shows that the state space is a 2-Dimensional box with each observation being a vector of two values. Also indicated here is the action space, comprising of three discrete actions; full throttle right, full throttle left and finally to do nothing.



```
[4] print(env.observation_space.low)

    print(env.observation_space.high) #

[-1.2  -0.07]
[0.6  0.07]
```

Figure 2: Range of Position and Velocity States.

From **Figure 2** it is observed that the position can range from -1.2 to 0.6 (first element) while the cart's velocity can range from -0.07 to 0.07 (second element). At this point the environment is represented by a continuous space in these state ranges, meaning that there are infinitely many state-action pairs (there can be many decimal points) which is an issue for the Q-learning algorithm. In order for the algorithm to converge it is a requirement that all state-action pairs are visited a sufficiently large number of times, impossible given the decimal precision of the continuous space. This issue was overcome by discretizing the state-space.

A Qlearning function is first developed, and the size of the discretized state space is determined. The Q-table is initialized by setting all elements to random numbers close to zero. Lists are then initialized so that we can later track the rewards, followed by the definition of epsilon reduction. Epsilon (ϵ) is a parameter specified between 0 and 1 that describes the rate of exploration of our agent in the state-action space. The higher it is, the more likely it is to carry out a random action to learn the results. To successfully learn the **RL** algorithm it is in general good practice to start with a large epsilon (epsilon-greedy strategy) so that it can learn information about the state space at the beginning when Q-table values are close to zero (unknown). Its reduction is important as over time the agent learns more and more information about the state space that it can exploit. This is completed in the algorithm by decreasing the parameter linearly over the amount of episodes.

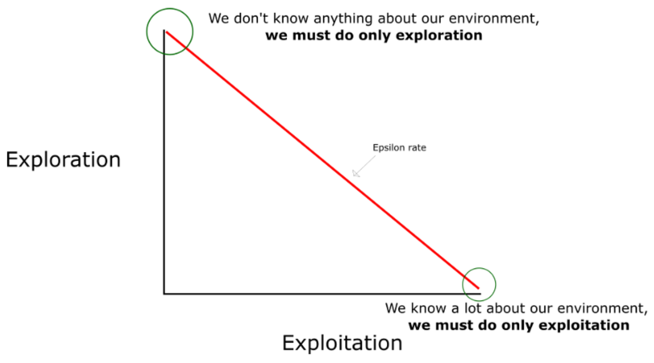

We don't know anything about our environment,
we must do only exploration
Exploration
Epsilon rate
We know a lot about our environment,
we must do only exploitation
Exploitation

Figure 3: Representation of ϵ reduction[3]

The Q-learning algorithm is then written in the code under a 'for' loop over the episodes, starting by initializing the rewards and resetting the state prior to learning. The 'Mountain Car' environment provided by Open AI Gym reached a terminal state after 200 movements had been completed by default (if it had not reached the goal by then). This could be adjusted however to investigate results when this terminal state was reached after a greater number of movements, giving the algorithm more freedom to explore a given policy in a given episode.

Discretization of the state space is then implemented. The position element of the state-space is rounded to the nearest 0.1 and the velocity element is rounded to the nearest 0.01. These values are multiplied by 10 and 130 respectively. This is in order to achieve a square Q-table (Please see) for the asymmetric amount of discretized values between the two state-elements. After doing this, we reduce the number of state-action pairs to $19x19x3 = 1083$, making the above requirement achievable. Another Q-learning function was written to investigate the effect of increasing the amount of state-action pairs, now multiplying the discretized states by 100 and 1300 respectively, corresponding to 108300 state-action pairs. Naturally these trials require far more episodes to become accurate as it must visit all these states a sufficiently large number of times.

A 'while' loop is then written for after initialization and discretization has been completed. The environment of the final episodes (most accurate) can be rendered using *env.render*, however this was generally omitted from the code as it was not necessary to investigate the the efficiency of our implementation where we can just look at the average rewards in the output plus the final Q-table.

The next action is determined according to the ϵ value at that iteration. A random number is generated and if it is greater than $1 - \epsilon$ then it will choose a random action (ie. explore). Otherwise it will choose the optimal action depending on the Q-table at that particular point. The next state and reward are then obtained by moving a step according to the action. These consecutive states must also be discretized under the same previous conditions and following this the condition for the terminal state is coded, where the car achieves a position of ≥ 0.5. If this is the case that the Q-value value representing the state-action pair is written as the reward. Otherwise, the Q-value is updated according to the Bellman equation:

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma maxQ(s',a') - Q(s,a)] \quad (1)$$

The total reward is then updated with the reward achieved at that iteration. The state is to be used for the next iteration is also updated with the current state.

Epsilon is reduced by the reduction value previous defined if it has not reached the minimum epsilon (which should only be reached by the final episode). The total rewards are appended to a list and these are averaged over a specified amount of steps/moves (10 for most trials). The average reward for a given amount of steps are then printed, and these are appended to an 'episode' list (used for plotting when running three trials for the same configuration and obtaining an average over the amount of episodes).

To visualize how the agent is learning, a function was written that plots the scores and their rolling mean for a specified window. The moving average calculates a series of averages for different subsets of the full data and so reduces the impact of noise.

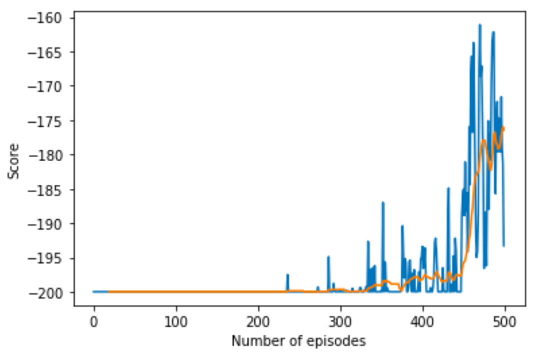
Finally a function was written to plot the final Q-table for a given configuration. This function was adapted from a solution found on GitHub[4]. It outputs a heat map visualization of the max Q-value for each state-action pair that has been explored in the Q-learning algorithm, and includes in each entry in the table a score value for that particular state.

2 Results and Analysis

Various configurations were for the Q-learning algorithm were implemented in this investigation. Firstly, various configurations were run over 5000 episodes to how various parameters affected the outcome of the agent. After this investigations were made into a increasing the size of the state-space, increasing the number of episodes and by averaging configurations over three different trials to investigate consistency. The value for minimum ϵ was retained at 0, as by the time training finishes, the agent has learned the maximum amount of information that it can.

2.1 Base Configurations of 5000 Episodes

Using relatively standard parameters as seen in **Figure 4** an intuitive Q-table is observed. The red boxes on the outskirts are in fact unexplored states by our agent. Generally this is due to the fact that these state-action


(a) Score with rolling mean Vs Number of Episodes/10

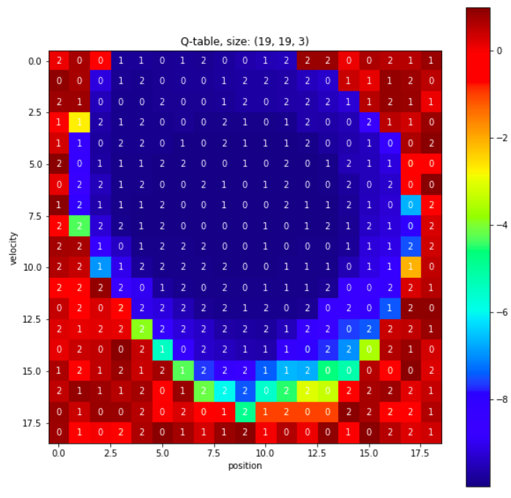
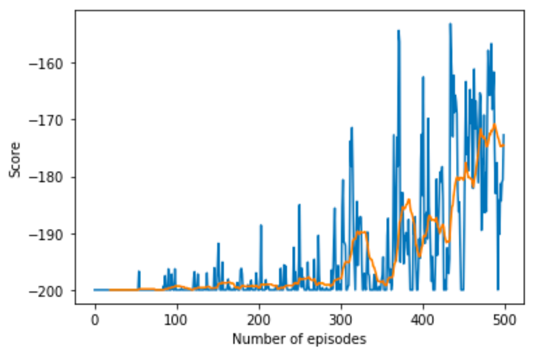

(b) Q-Table

Figure 4: $\alpha = 0.2, \gamma = 0.9, \epsilon = 0.8, Episodes = 5000$, Lowest Average Reward = -167.5

pairs are unexplorable given the configuration. Starting in the middle of the table, ie. 0 position and velocity leads to the lowest scores. A trajectory can be traced on the outside of the blue colored circle, where lower scores are expected due to a more ideal position/velocity. From the score plot we can see that the agent starts learning more rapidly after approximately 4500 episodes. At this point it has explored much of the state-action pairs and by exploiting more is able to achieve higher scores.

Using smaller ϵ values as observed in **Figure 5** and **Figure 6**, it is observed that less of the space is explored. As the the agent explores less and tries to use accumulated information earlier, it starts learning more rapidly sooner using a smaller amount of ways to achieve the goal, as sign from the moving average plots. However, as seen in **Figure 6** the highest score of these initial trials were achieved close to the end of training even though it may be focused on a smaller amount of ways to achieve this score.


(a) Score with rolling mean Vs Number of Episodes/10

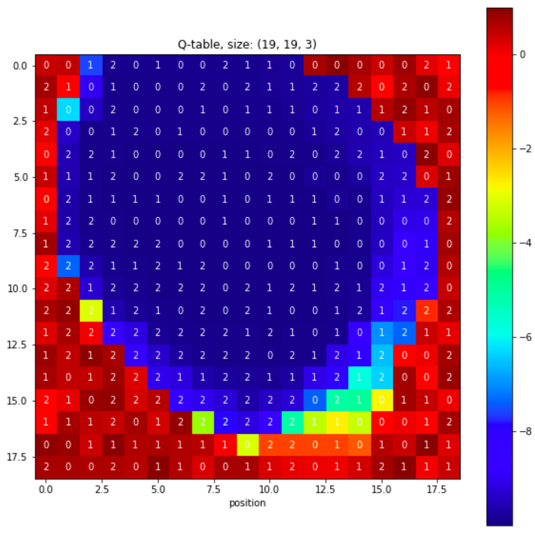
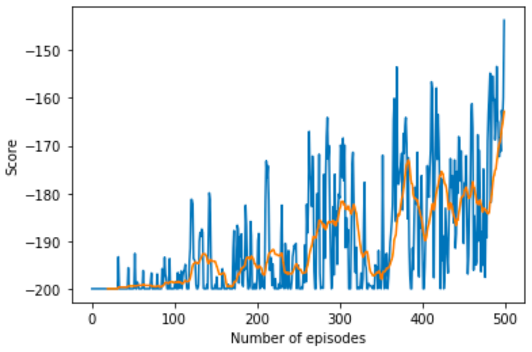
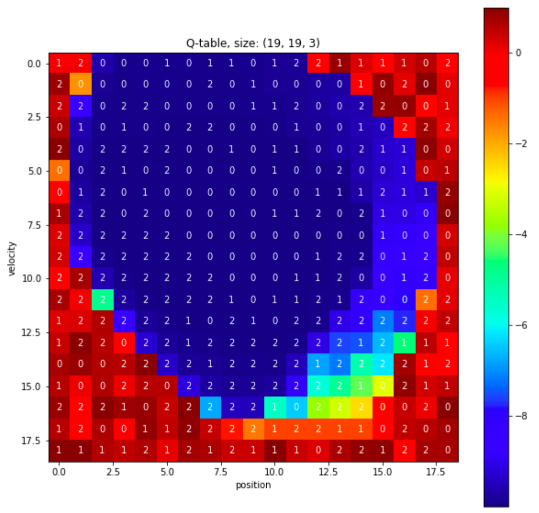

(b) Q-Table

Figure 5: $\alpha = 0.2, \gamma = 0.9, \epsilon = 0.4, Episodes = 5000$



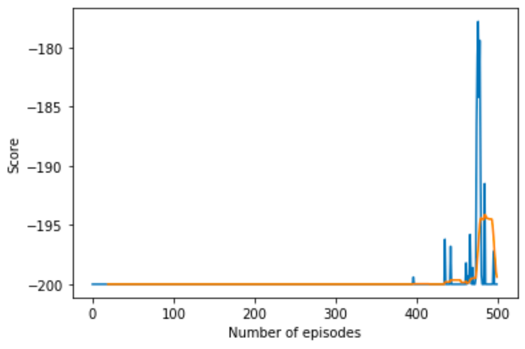
(a) Score with rolling mean Vs Number of Episodes/10



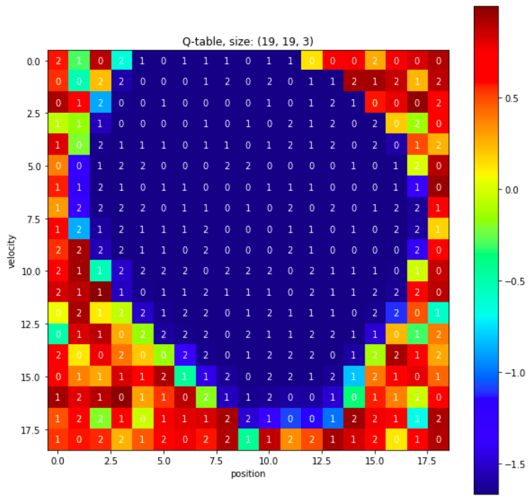
(b) Q-Table

Figure 6: $\alpha = 0.2, \gamma = 0.9, \epsilon = 0.2, Episodes = 5000$

Using smaller γ values as observed in **Figure 7** and **Figure 8**, poorer results are achieved over 5000 episodes. The lower the discount, the less the agent is influenced by distant rewards. As a result it does not achieve as well as previous configurations as it is struggles to obtain insight into moves at much further time steps. It also poses a representation info as the reward scale is now far from linear. With a low γ the distance between rewards at consecutive time steps flattens out exponentially, and so the Q-tables display much less variation in colour (all dark blue in the middle).



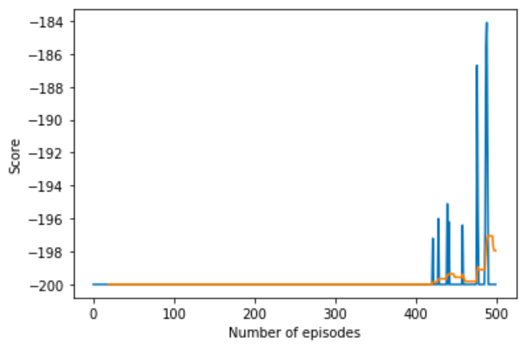
(a) Score with rolling mean Vs Number of Episodes/10



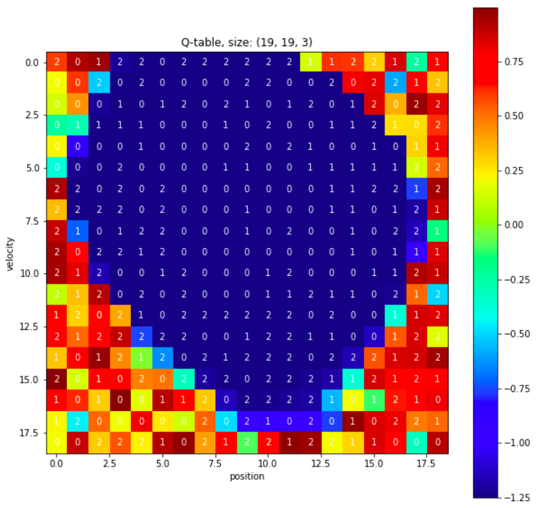
(b) Q-Table

Figure 7: $\alpha = 0.2, \gamma = 0.4, \epsilon = 0.9, Episodes = 5000$

The configuration observed in **Figure 9** displays promising results, with a sharp spike in the moving average for the last 1000 episodes. The final Q-table also indicates a well-trained agent that has explored much of the state space sufficiently. If considering training over more episodes, parameters similar to the ones used in this configuration are desirable.

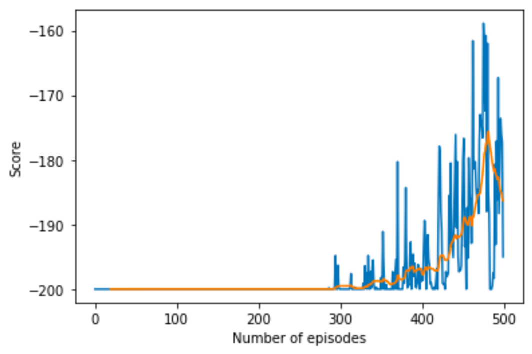


(a) Score with rolling mean Vs Number of Episodes/10

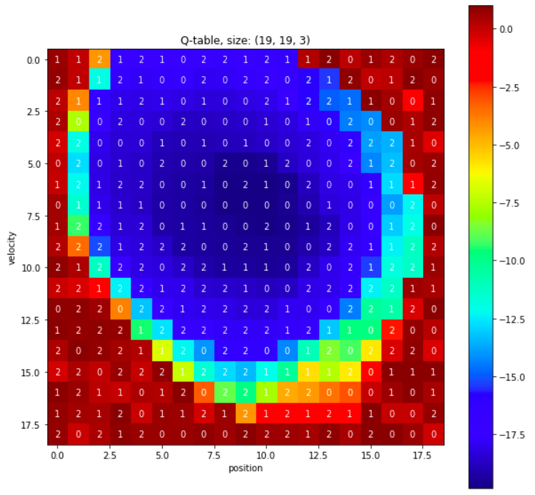


(b) Q-Table

Figure 8: $\alpha = 0.2, \gamma = 0.2, \epsilon = 0.9, Episodes = 5000$



(a) Score with rolling mean Vs Number of Episodes/10

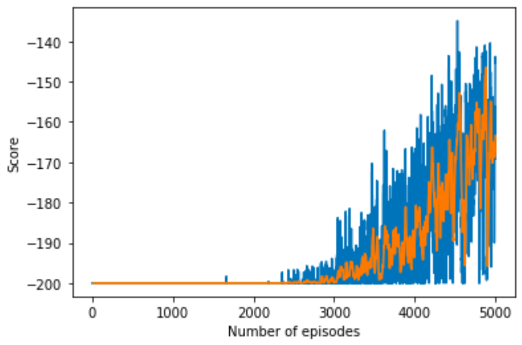


(b) Q-Table

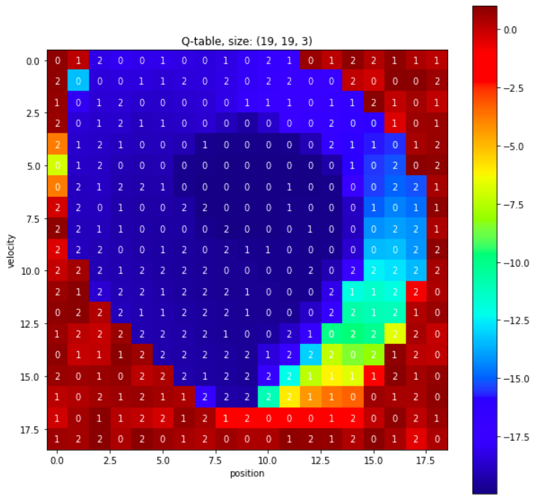
Figure 9: $\alpha = 0.15, \gamma = 0.95, \epsilon = 0.95, Episodes = 5000$

2.2 Increased No. of Episodes

A similar configuration to **Figure 9** was used to see how it performed when trained over 50000 episodes and using a smaller learning rate of 0.1 (due to more episodes). More accurate results were achieved here as a result, and it is observed that by 50000 episodes that the agent is learning rapidly. For future investigations, training could be done over much more episodes (eg. 500,000 as in next stage) but over this size state-space. The Q-table however does not actually look as good as the previous test in **Figure 9**, with states to the left of centre receiving lower scores. This suggests that that positions starting up the left hill are not optimal, however the ideal trajectories occurring at the right centre of the Q-table are preserved.



(a) Score with rolling mean Vs Number of Episodes/10

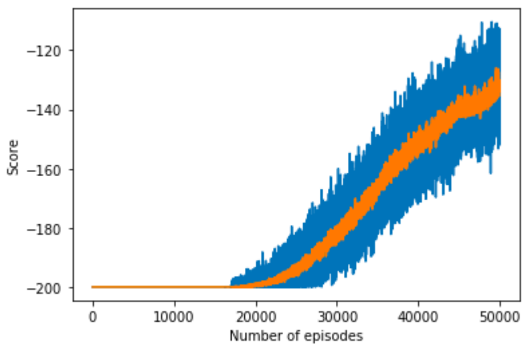


(b) Q-Table

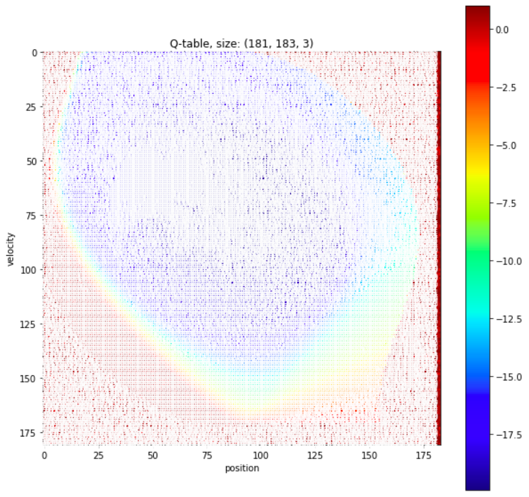
Figure 10: $\alpha = 0.1, \gamma = 0.95, \epsilon = 0.95, Episodes = 50000$

2.3 Increased State-Space

When increasing the size of the state space by a factor of 1000 (ie. $190x190x3 = 108300$) some interesting results were obtained. For this configuration similar parameters to those seen in **Figure 9** were used. In this case we can see that the Q-table is broken up into a much larger amount of very small boxes where the action values are not distinguishable. The image is also quite saturated. However the highest score was achieved here out of all configurations, and looking closely quite detailed trajectories can be observed at the right and bottom parts of the table for ideal state-action pairs. This configuration provided the highest scores of all configurations (as seen in **Figure 11**, which is encouraging due to the much higher number of states involved, albeit that it was run over a much larger amount of episodes. The plot of the scores indicates a steady increase of the moving average.



(a) Score with rolling mean Vs Number of Episodes/10

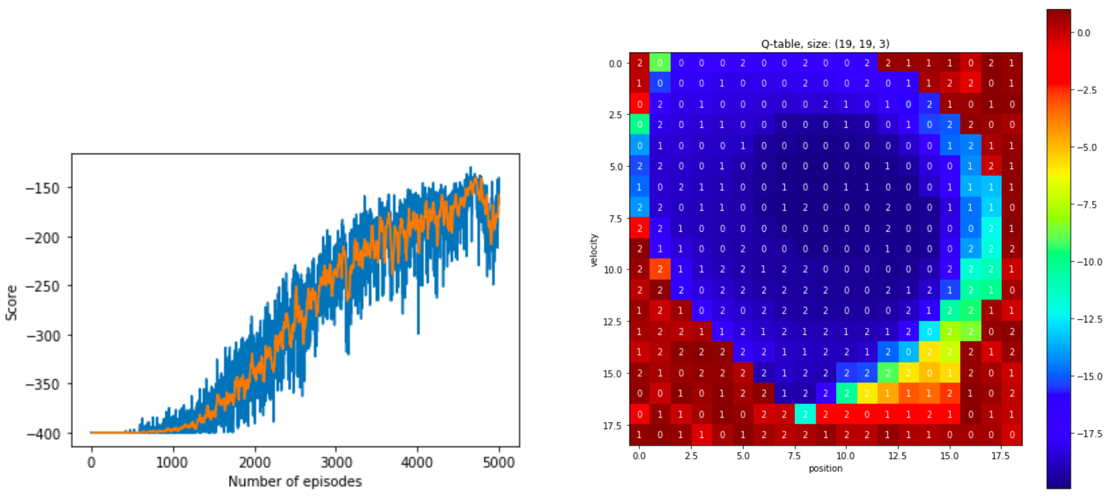


(b) Q-Table

Figure 11: $\alpha = 0.1, \gamma = 0.95, \epsilon = 0.95, Episodes = 500000$

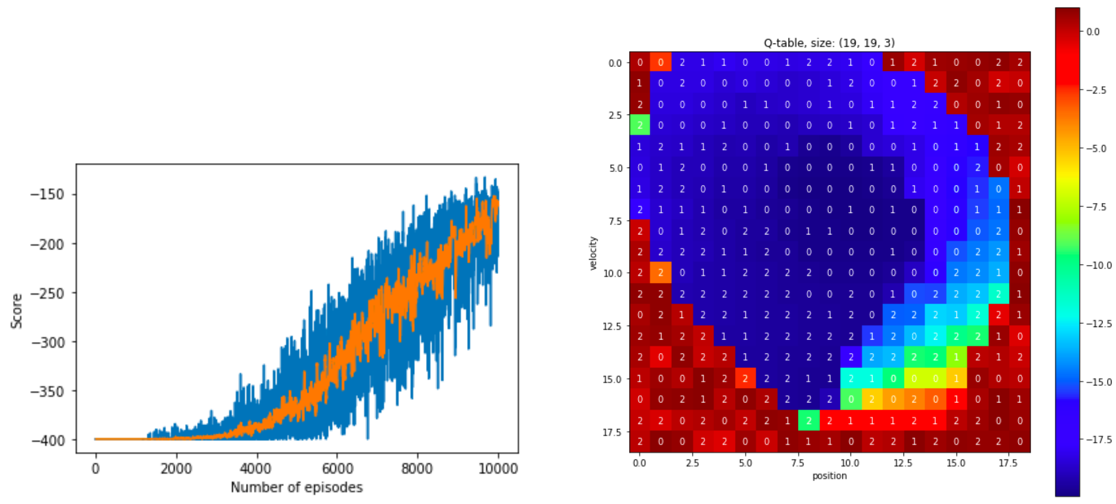
2.4 Increased Max-movements (400)

For this final experiment the maximum number of movements before reaching the terminal state was changed from the default 200 to 400, giving the agent more freedom to learn before having to reset state. When choosing a smaller learning rate of 0.05 a lower variance of scores and a more defined Q-table is obtained. A similar trend in the plot of scores is observed. However in **Figure 12** is is observed that convergence occurs much faster than in **Figure 13**, which is interesting given the much smaller learning rate. As a result, for future investigations, configurations with smaller α values could be investigated as this was the only occurrence of using one this small. Although the run-time over the same number of episodes is longer, high scores and detailed Q-tables are obtained when increasing the maximum number of movements.



(a) Score with rolling mean Vs Number of Episodes/10 (b) Q-Table

Figure 12: $\alpha = 0.05, \gamma = 0.95, \epsilon = 0.95, Episodes = 50000$



(a) Score with rolling mean Vs Number of Episodes/10 (b) Q-Table

Figure 13: $\alpha = 0.05, \gamma = 0.95, \epsilon = 0.95, Episodes = 50000$

3 Conclusion

This investigation has shown that Q-learning is a very effective algorithm for reinforcement learning. In all above configurations, the agent was successfully trained to achieve the goal of 'Mountain Car', although the speed of convergence does vary considerably. Even when low ϵ or γ values were used, hindering the ability to explore and to make more concrete decisions based on future rewards respectively, the agent still managed to reach the target in a relatively low number of episodes. In any case, however, there were certain configurations that performed far better, those being γ and starting ϵ values between 0.9 and 1. For future work, more investigations would be made into a small learning rate parameter (order of 0.01), while also more episode iterations would be run for configurations similar to that in **2.2**

4 References

1. https://gym.openai.com/
2. https://gist.github.com/gkhayes/3d154e0505e31d6367be22ed3da2e955
3. https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/
4. https://github.com/udacity/deep-reinforcement-learning/blob/master/discretization/Discretization_Solution.ipynb