Advanced Concepts of Machine Learning: Reinforcement Learning

Kirill Shcherbakov (i6248484), Shashank Subramanya(i6257859)

November 25, 2020

1 Introduction and general information

All experiments are performed on local machine.

2 Software description and how to use it

2.1 Prerequisites

The following dependencies must be installed to run the code:

1. Python 3.x
2. NumPy 1.18.5 and Matplotlib 3.2.2
3. Seaborn
4. Gym 0.7.4

2.2 How to launch the code?

- Firstly, you should open the corresponding python script in the Python IDE or text editor of your choice locally on your computer.
- In the script the code is arranged in a sequential way. Run the entire code and then wait for the result to run.
- The following training parameters are available for modification: the learning rate, discount, epsilon and episodes.

3 Implementation details

3.1 Approach utilized for problem solving

The Q-learning algorithm was chosen to solve the Mountain car problem for the following reasons:

- It is easier to learn the optimum Q value in a model free environment where the full MDP with transition probabilities required for calculating the state value function is not available
- The state value function is obtained by picking the action that maximizes Q value for each state and thus it can be visualized
- It enabled us to maintain a table of Q values for each state action pair and update it in each step. A function approximation to calculate Q values was not utilized as the problem could be solved by iterating through a finite state space

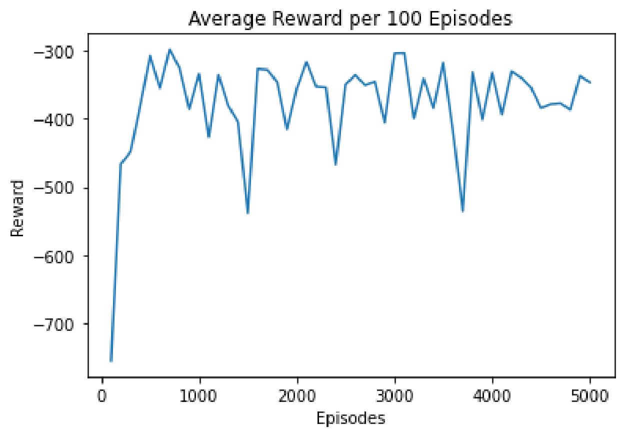
3.2 Handling continuous state space

As mentioned previously, the Mountain car problem could be solved by iterating through a table lookup of Q values and hence we discretized the continuous state space. Position from -1.2 to 0.6 was represented through 19 states separated by a step size of 0.1 and speed from -0.07 to 0.07 was represented by 15 states split by a step size of 0.01. Each state had 3 actions, full throttle ahead, reverse, or no action at all. Thus, there were 285 (19*15) states and 855 state action values in our state space.

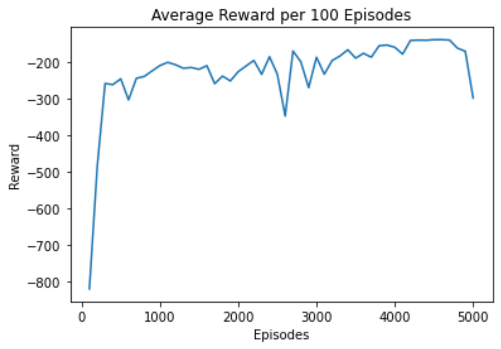
4 Experiments and results

To get the best visualization of the state space, the effect of the parameters discount, learning rate, and epsilon was observed on the total reward from each episode across multiple episodes of the Q learning algorithm.

The first set of experiments were run with no epsilon decay and discount factor as 1. Figure 1 shows the plot of reward from each episode averaged over a 100 episodes across 5000 episodes. In figure (a) we can see that algorithm is unable to learn a solution without decaying epsilon as the randomness introduced by epsilon is hindering the learned Q value from converging to the optimal one. In figure (b) epsilon decay was introduced but rewards were not discounted and the algorithm failed to converge again. Since the solution involves moving back up the hill to gain momentum, discounting is needed to ensure that algorithm is able to exploit this policy that is learned through exploration.



(a)

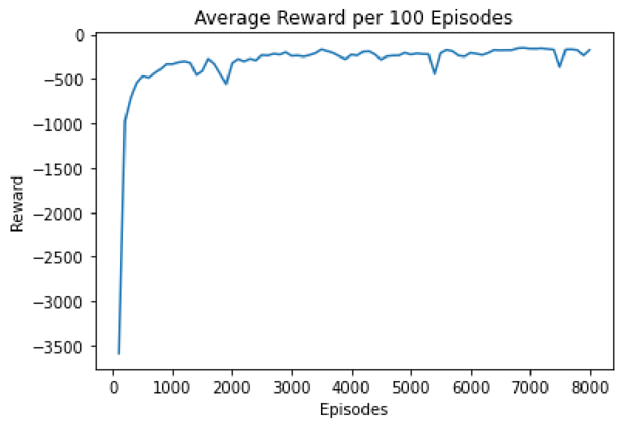


(b)

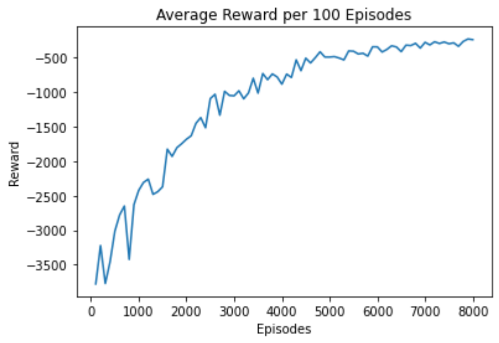
Figure 1: Reward averaged over each 100 episodes (a) ϵ=0.2 with no decay (b) γ=1

From the above experiments we concluded that the reward had to be discounted and ϵ had to be decayed for convergence of the Q learning algorithm. So, we set γ as 0.9 and reduced ϵ by a factor of the number of total episodes after each episode. Next, we experimented with the learning rate and ϵ values. Figure 2 shows the outcome for low α and high ϵ values. In figure (a) the algorithm improved slowly and had an average reward of only around -200 with a low learning rate even after 8000 episodes. In figure (b) we can see that a high ϵ value induces greater randomness in choice of policy and thus it takes a longer time to converge to optimum Q. In our example, it couldn't converge even after 8000 episodes.

Finally, the best performing parameters were α=0.2, γ=0.9, and ϵ=0.1. Figure 3 shows the total reward per episode averaged over each 100 episodes with these parameters. The algorithm learnt Q values that could consistently find a solution to the problem in about 140 steps after 5000 episodes. By turning ϵ-greedy exploration off after reaching a reward of -160, we could ensure that the Q value was further greedily maximized in the last few episodes.



(a)



(b)

Figure 2: Reward averaged over each 100 episodes (a) α=0.01 (b) ϵ=0.8

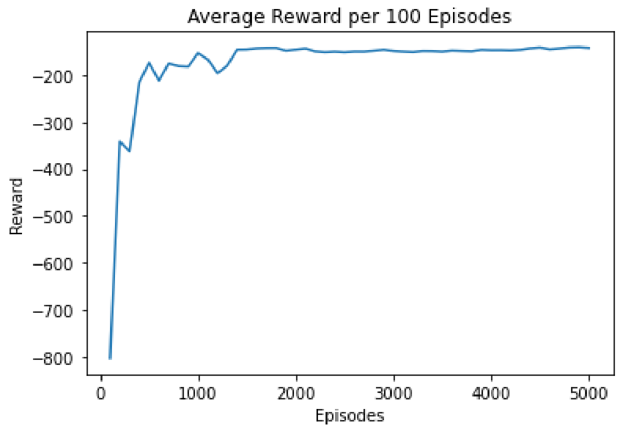


Figure 3: Reward averaged over each 100 episodes α=0.2, γ=0.9, and ϵ=0.1

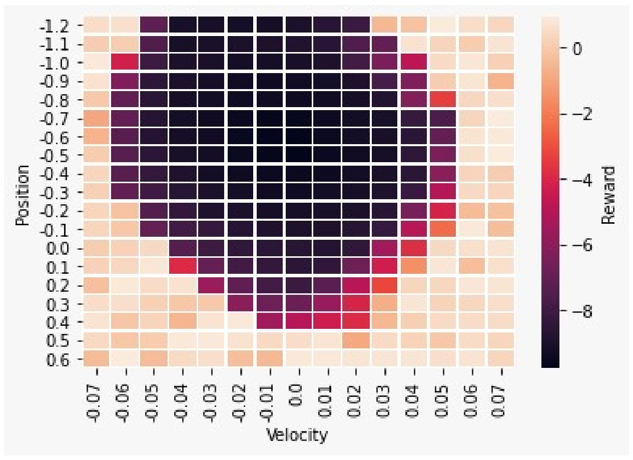


Figure 4: Value Function by state

Figure 4 shows the value function visualization from the Q values learnt from the above mentioned parameters. The value function was calculated as the maximum Q value among the 3 actions for each state. We can observe that the optimum policy is one that involves having a high positive velocity when position is less than 0 and having high positive and negative velocities when position is greater than 0 depending on whether the car is making its first ascent or descending from the top to gain momentum.

Finally, a more optimal solution could be obtained by treating the action space as continuous and representing it as a feature space of position and velocity. We would then need to find a value function approximation to predict the optimal value function which could be achieved using a Neural Network.

Also in the future work, it is worth to play more with discretization amount to obtain an even better visualisation of the state value function.

Due to the fact that no one in our team had any previous experience with reinforcement learning, this laboratory work was of average complexity for us. It took about one day to study and select an algorithm for this problem. Also, it took about two days to implement the algorithm in Python and to get acquainted with OpenAI gym and install the necessary dependencies. It took about a day to experiment with hyperparameters and the environment and get the algorithm to learn something useful, and it was quite time-consuming for us.