Performance of SARSA and Q-Learning on Cartpole and MountainCar

Rohan Akut
260954888

1 MDP for Cartpole:

We have to balance a pole vertically on a cart in this problem. The entire description of the problem can be found in [1]

1.1 My understanding of the problem:

The problem supplies us with four states, namely: cart position, cart velocity, pole angle and pole angular velocity. These states are continuous, and the range of these values can be found in [2]. However, to formulate an MDP, we need discrete states. Hence I have converted the states from their continuous space to discrete space. The code to discretise the states has been referred from [3]

1.2 States:

As mentioned earlier, openAI supplies us with 4 states[2].However, after referring to the explanation in [4], I understood that the two most important states in this problem are pole angle and pole angular velocity. As mentioned in [4] these two states are primarily responsible for maintaining a vertical position on the pole. Hence, I have just considered these two states to tackle this problem. Initially, I had just considered pole angle and ran the two algorithms. Post this, I have considered both the states(pole angle and pole angular velocity) and ran Sarsa and Q-learning on the two states.

1.3 Actions:

As mentioned in [2], the two possible actions are applying force to the left of the cart or right of the cart. I have not changed the actions, and the same actions are used for solving the problem.

1.4 Reward:

By default, openAI provides a reward of +1 for every timestep the pole stays upright[2]. However, I believe this reward function does not reflect the problem well. In my opinion, a reward should be provided for every correct action, and a penalty should be given for every wrong action. Hence I have used pole angle for my reward calculation. For every action taken, I check if the pole angle is greater than the pole angle in the previous timestamp. If the pole angle has increased, then it means that an incorrect action has been taken. Hence I punish the algorithm. The same applies when the pole angle decreases compared to the previous timestep. If the pole angle decreases, the algorithm is rewarded positively for taking the correct action. The idea for this custom reward has been referred from [5]

I have also added a huge penalty if the algorithm terminates before 30 timesteps. It is to be noted that the idea of adding this huge penalty was referred from [6].

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

1.5 Declarations to preserve academic integrity:

As mentioned previously, I have referred to multiple resources for formulating my MDP. In this section, I would like to summarise all the sources to ensure that the appropriate authors have been cited.

1. **States:** The idea for using just two states has been referred from [4] and the method for the discretisation of states has been referred from [3]
2. **Actions:**The possible actions have not been modified. Hence the default actions provided in [2] have been used.
3. **Reward:** The reward function has been designed by referring to two sources. The idea of penalising according to pole angle has been referred from [5].Moreover, the idea of adding a huge negative reward when the algorithm finishes before 30 timesteps has been referred from [6].

I have also added the appropriate citations in the code file to preserve academic integrity.

2 MDP for MountainCar:

In this problem, we have to help the car reach the top of a mountain by using the laws of physics. The entire problem description can be found at [7]. In this section, I will explain the MDP formulation for solving this problem

2.1 States:

As mentioned in [8], the two states are: car position and car velocity. To solve this problem I have used both of these states. These states are available as continuous variables. Hence to discretise these states, I have referred to the code mentioned in [9].

2.2 Actions:

This problem has three possible actions: accelerate to the right, accelerate to the left, no acceleration[7]. I have used the default actions supplied by openAI for solving this problem.

2.3 Reward:

By default openAI gives a reward of -1 for every timestep where the car doesn't reach its goal. It provides a reward of 0 when the target position is achieved. However, I believe that this reward function would not help the car in reaching the target position. Hence I referred [5] to design my custom reward function. As mentioned in [5] I have given a positive reward every time the car gets closer to the goal. This means that instead of just setting a positive reward for the final target, the car would receive a reward if it climbs a certain portion of the hill. This idea of giving rewards according to car position is referred from [5]. I have given a pictorial representation of this custom reward function in Fig 1. The Fig 1 is not exactly to scale. However, it has been added to give a graphical representation of the custom reward chosen for this problem

2.4 Declarations to preserve academic integrity:

As mentioned previously, I have referred to multiple resources for formulating my MDP for Mountain Car Problem. In this section, I would like to summarise all the sources to ensure that the appropriate authors have been cited.

1. **States:** I have used the default states which are supplied by OpenAI[7]. However, these states have been disretised and the logic for discretisation has been refered form [9]

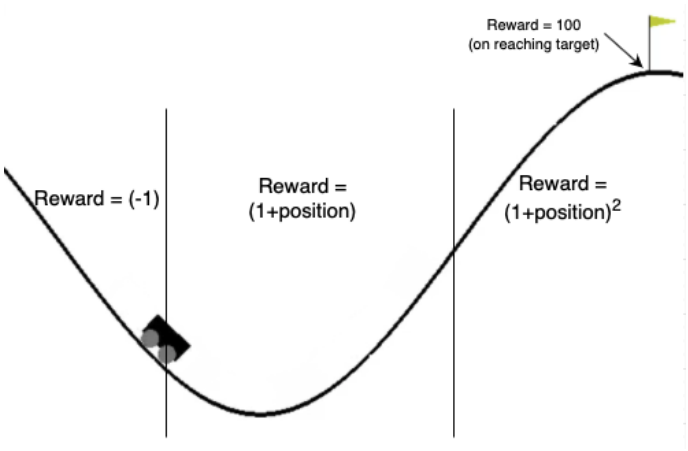


Figure 1: Custom Reward function for MountainCar[5]

2. **Actions:**The possible actions have not been modified. Hence the default actions provided in [7] have been used
3. **Reward:** The custom reward function has been referred from [5]. The pictorial representation of the reward function can be found in Fig 1.

3 Results for CartPole:

This section will describe the results that I obtained on running SARSA and Q-learning algorithms on CartPole problem. Before moving on to the MDP mentioned in Section 1.2 I have first created a simple MDP which just includes pole angle as a state. I have used this MDP to get a hang of openAI and understand the problem in a better sense. Hence initially, I will describe my findings on a Single State MDP for SARSA and Q-learning, and then I will move to the Two-State MDP mentioned in Section. 1.2

3.1 Single State MDP:

I decided to run SARSA and Q-learning on a single state MDP because of the following reasons:

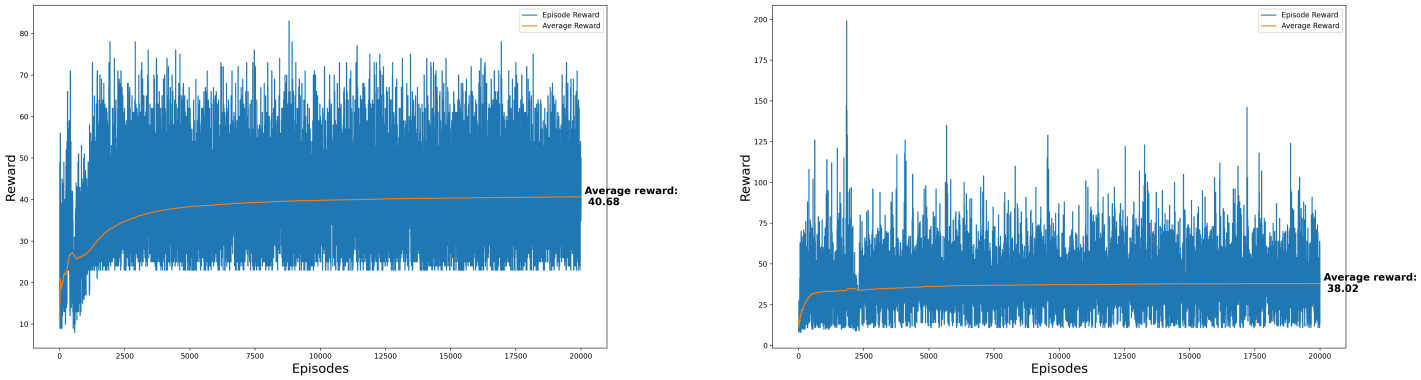
1. For this problem I was converting the continuous states into discrete values. However, I did not know the ideal values for discretisation. Hence I decided to test the discretisation values on a simple single-state MDP.
2. Even though the reward function has been refered from [5], I wanted to check if the reward function actually helps the models in converging faster

Hence I would like to mention that the purpose of Single-State MDP was **not to solve the problem, but to understand the system and the ideal discretisation values for the CartPole problem**.

3.1.1 Results for Single-State MDP:

In this section I will describe the results obtained after running SARSA and Q-learning on Single State Cartpole

3.1.1.1. SARSA: As mentioned in 3.1 the only state used in this MDP is pole angle. Using this as my state value I have tried to train SARSA. The SARSA model was trained for 20,000 episodes with a learning rate of 0.2 and discount factor of 0.7. The results of the training are as shown in Fig 2 The discussion for this figure has been provided in Section 3.1.2



(a) Single State SARSA

(b) Single State Q-learning

Figure 2: Performance of algorithms on Single State CartPole with learning rate = 0.2 and discount factor = 0.7

3.1.1.2. Q-learning: Similar to SARSA I have trained the Q-learning algorithm on CartPole problem for 20,000 episodes with a learning rate of 0.2 and discount factor of 0.7. The results for the training are as shown in Fig 2 Similar to SARSA, the discussion for Fig 2 has been provided in Section 3.1.2

3.1.2 Discussion for Single State MDP

In this section, I will provide an analysis of Fig 2. As mentioned in Section 3.1 the main reason to run a Single state MDP was to understand the problem properly. Hence after running both the algorithms, the following were my conclusions.

1. It is clear from Fig 2 that a single state MDP is not enough to solve the CartPole problem. However, it does show that both the models(SARSA and Q-learning) are learning and are converging to a maximum. This is encouraging as this proves that the implementation of these algorithms is, in fact, correct. With more information, these algorithms could solve the CartPole problem.
2. While designing the MDP, the reward function is crucial to help the model converge faster. If I had not added the custom reward function mentioned in Section 1.4 then both the algorithms do not converge properly. The average reward for both of the algorithms is around 20 as compared to 40 as visible in Fig 2. This shows that the implementation of my reward function is correct and it helps the algorithms in convering faster.
3. The discretisation level for this problem is very important. For example, if I do a coarse discretisation, I observed that SARSA and Q-learning never learnt anything worthwhile, and the average reward was less than 10. The same happened when the state discretisation was very fine. In this case these algorithms had too many options to choose from, and hence it got confused. Hence, it is imperative to strike a proper balance between the amount of discretisation to help these algorithms learn properly. It is to be noted that I have referred to [9], to find the ideal values for state discretisation. The same has been mentioned in the code as well.

Thus, after having an in-depth idea of the problem statement and understanding the ideal values of discretisation and reward functions, I shifted my focus from Single state MDP to a two-state MDP. In the following section, I will describe the performance of the 2-state MDP mentioned. The formulation of this MDP has been mentioned in Section 1.2

3.2 Two State MDP:

Building upon the information gained in Section 3.1.2, I have implemented a two-state MDP and have run SARSA and Q-learning on this MDP. In this section, I would give a detailed analysis of the performance of SARSA and Q-learning on this MDP

3.2.1 SARSA:

As mentioned in Section 3.1.2, the ideal reward function and the function approximation values have been determined. However, we still need to find the ideal values of learning rate and discount factor to solve CartPole problem successfully. Hence in this section, I would first present an analysis of the performance of SARSA for different learning rates and discount factors. Then, once these hyperparameters are set, I would run the SARSA algorithm on the CartPole problem.

3.2.1.1. Determining Learning Rate and Discount Factor: In order to determine these hyperparameters I have run SARSA on CartPole problem for 5000 episodes with the values of learning rate varying from 0.1 to 1.0. The same has been done for discount factor. The results after running the SARSA algorithm for different hyperparameters can be found in Fig 3. As visible in Fig 3 SARSA

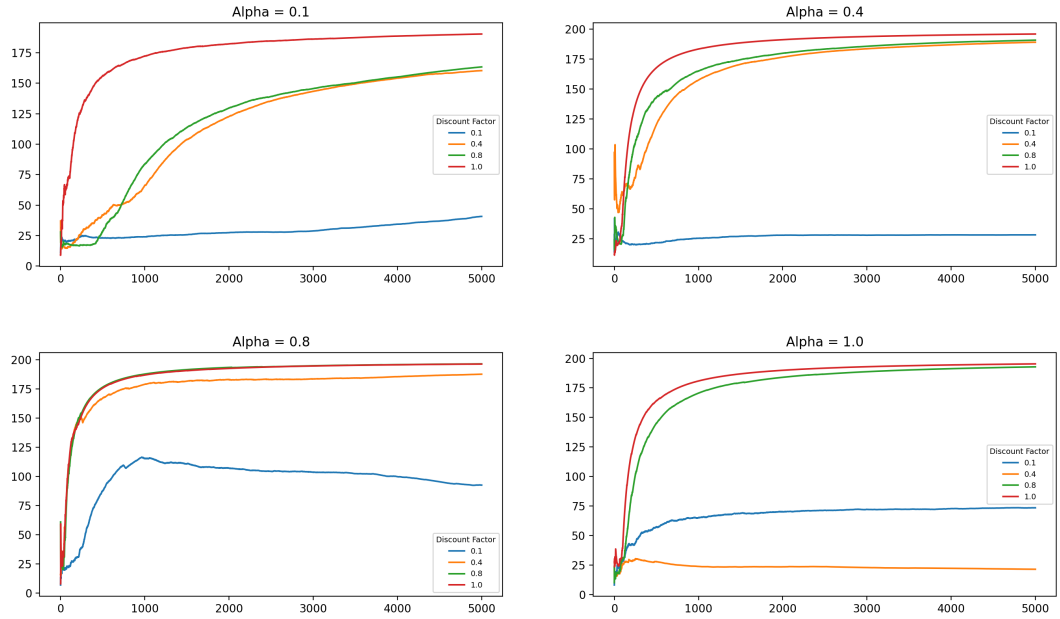


Figure 3: SARSA performance against different learning rate and discount factor values

gives a very high reward when learning rate = 0.8 and discount factor is either 0.8 or 1.0. Since the discount factor of 1.0 could put an overdependence on future values, I have chosen value of 0.8 for my learning rate and 0.8 for my discount factor.

3.2.1.2. Performance of 2 State SARSA: Since all the hyperparameters are fixed I have now run SARSA on Cartpole problem. the algorithm is run for 3000 episodes with learning rate and discount factor set to 0.8 as mentioned in Section 3.2.1.1. The performance of SARSA on Cartpole is as shown in Fig 4. As visible in Fig 4 the SARSA reaches the maximum reward(200) in less than 500 episodes. If I run this algorithm for a longer time, my algorithm would solve the problem. I will extend this analysis to the Q-learning algorithm in the following sections.

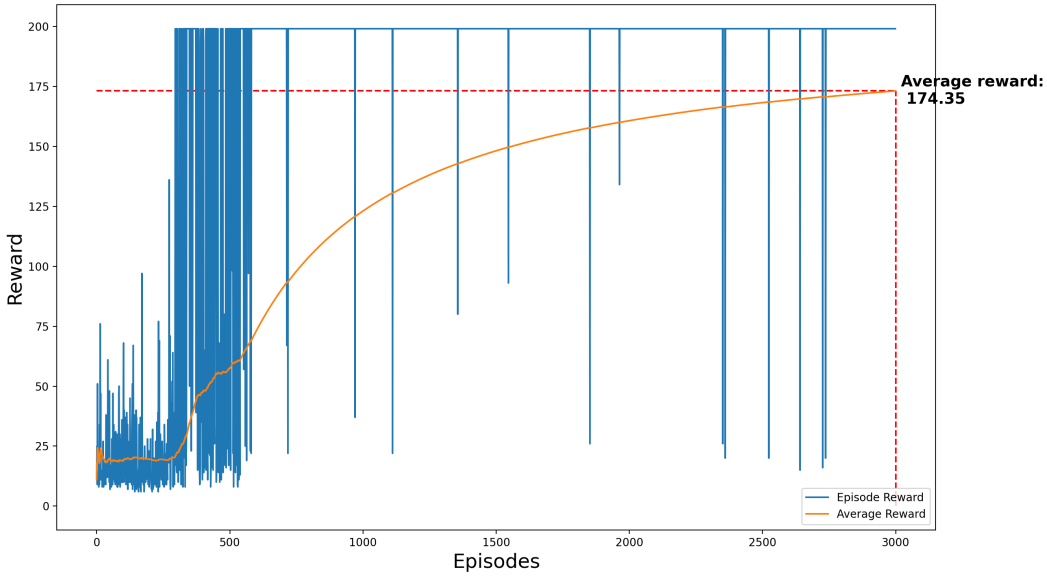


Figure 4: Two State SARSA with learning rate = 0.8 and discount factor = 0.8

3.2.2 Q-learning:

Similar to SARSA(Section 3.2.1) I have calculated the optimal values of learning rate and discount factor for Q-learning algorithm. Post that I have run the Q-learning algorithm on Cartpole and analysed the result.

3.2.2.1. Determining Learning Rate and Discount Factor: Similar to Section 3.2.1 I have varied the hyperparameter values(learning rate and discount factor) between 0.1 to 1.0. The performance of Q-learning on Cartpole is as mentioned in Fig 5. As visible in Fig 5 the learning rates of 0.4, 0.8 and 1.0 give a high reward. for the discount factor values of 0.4, 0.8 and 1.0. Since there is not much difference in the reward for the above mentioned hyperparameter values, I have randomly chosen the learning rate of 0.4 and discount factor of 0.8 for my problem.

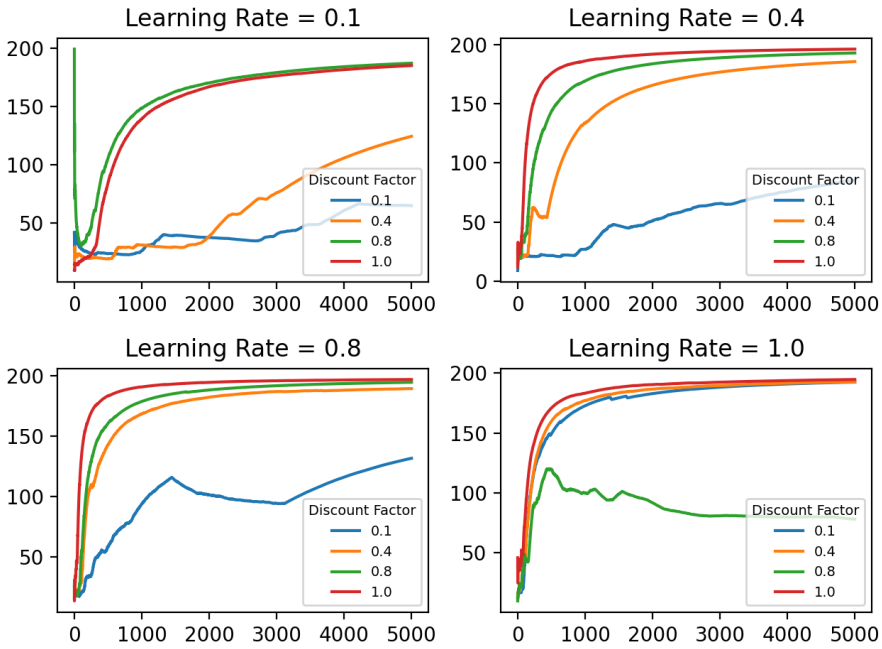


Figure 5: Q-learning performance against different learning rate and discount factor values

3.2.2.2. Performance of Q-learning on 2State CartPole: Since all the hyperparameters are fixed I have now run Q-learning on Cartpole problem. The algorithm is run for 3000 episodes with learning rate and discount factor set to 0.4 and 0.8 respectively. The performance of Q-learning on Cartpole is as shown in Fig 6. As visible in Fig 6 the Q-learning reaches the maximum reward(200) in less than

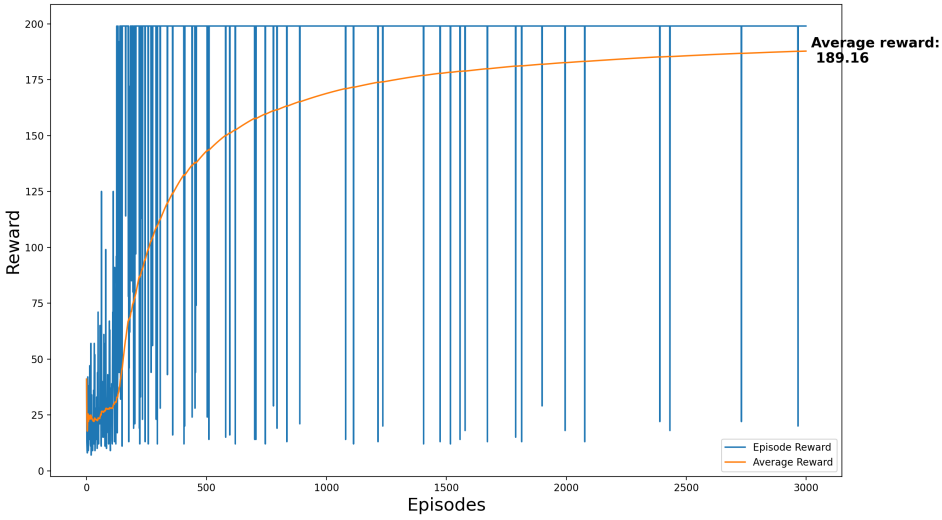


Figure 6: Two State Q-learning with learning rate = 0.4 and discount factor = 0.8

500 episodes. If I run this algorithm for a longer time, my algorithm would solve the problem.

Thus, in Section 3 I have initially determined the ideal reward function and the ideal discretisation values. Post that, I have determined the ideal values for hyperparameters and once all these values are set, I have run SARSA and Q-learning on the problem. I will perform a similar analysis on the MountainCar problem in the following section.

4 Results for MountainCar:

I have explained the MDP formulation for this problem in Section 2. In this section, I will discuss the performance of SARSA and Q-learning on the MountainCar problem using the MDP mentioned in Section 2.

Since I have not designed the reward function for this MDP and the discretisation has been referred from [5], I have not performed a Single State MDP analysis for the MountainCar problem. I have directly proceeded with a a 2-state analysis.

In this section, I will analyse the performance of SARSA and Q-learning on the MountainCar problem. As mentioned in Section 2, I have used a two-state system to evaluate this problem. In the following sections, I will describe the performance of both the algorithms on the 2-state MDP mentioned in Section 2 .

4.1 SARSA:

Similar to Section 3, I have first determined the ideal values of hyperparameters and then tested the performance of SARSA on MountainCar. Hence in the following sections, I will first determine the optimal hyperparameters and then check the performance of SARSA on MountainCar.

4.1.1 Determining Learning Rate and Discount Factor:

Similar to Section 3.2.1, the learning rate and discount factor are varied between 0.1 and 1.0. The algorithm has run for 5000 episodes, with every episode running for 1000 episodes. I have also performed an epsilon decay strategy to ensure that the agent explores the system before learning. The idea of epsilon decay has been referred from [9]. In order to ensure that I am choosing the optimal hyperparameter values, I have put an additional condition for termination. I have only terminated the training of the car if it reaches the terminal position more than five times. This ensures that only those hyperparameters are selected which learnt the optimal policy. The results can be found in Fig 7. From Fig 7 it is clear that when the discount factor value is high, the car reaches its goal five times and the training stops. These values are ideal values for training SARSA. In most cases, the car reaches the target(5 times) when the discount factor = 1.0. However, it is not common to set a discount factor of 1.0. Hence, I have chosen a discount factor of 0.95. For learning rate, the ideal learning rate could be either 0.8 or 1.0. In this case I have chosen 0.8 as my learning rate.

4.2 Performance of SARSA:

I have used the same algorithm that I had used for CartPole problem. As mentioned in previous section the hyperparameter values are set to 0.95(discount factor) and 0.8(learning rate). The termination condition for this algorithm is when the algorithm reaches the target position ten times. The SARSA algorithm is run for 10,000 episodes with each episode running for 1000 timesteps. The results obtained after running the SARSA model on MountainCar are shown in Fig 8. As visible in Fig 8, the SARSA algorithm reaches the target position five times in just 5000 episodes. Thus the algorithm is terminated before the total assigned episodes(10,000) are completed. This proves that my SARSA implementation solves the MountainCar problem.

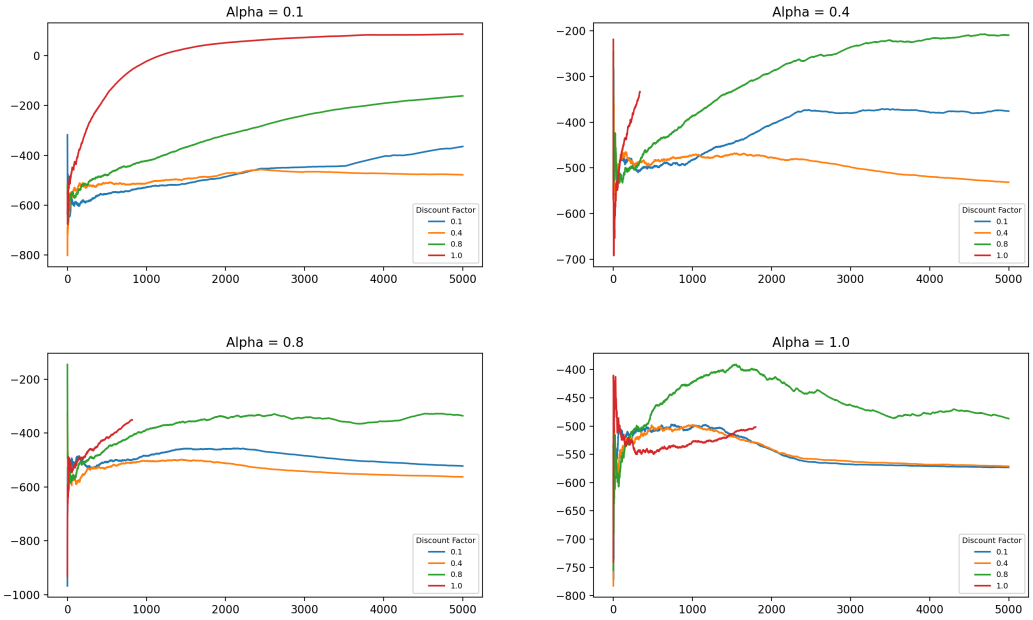


Figure 7: SARSA performance against different learning rate and discount factor values

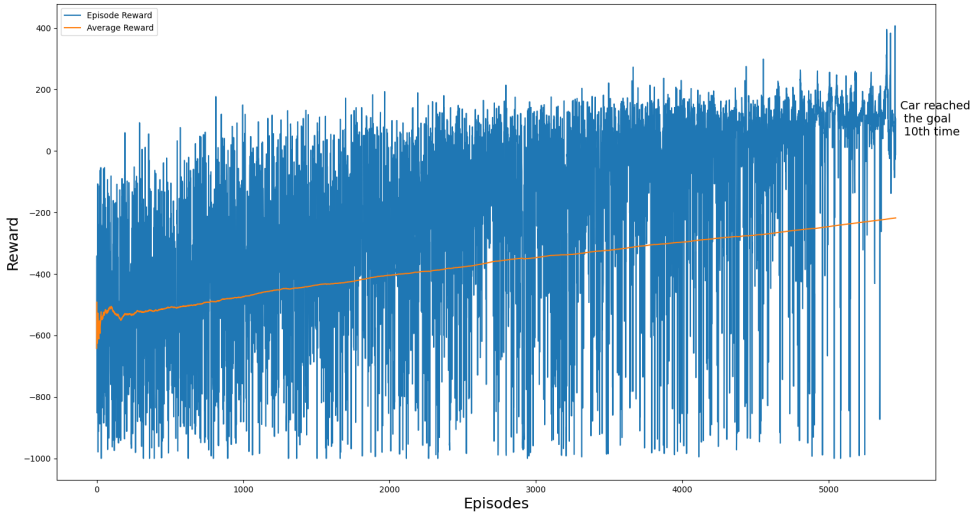


Figure 8: SARSA with learning rate = 0.8 and discount factor = 0.95

4.3 Q-learning:

In this section, I will discuss the performance of the Q-learning algorithm on the MountainCar problem. Similar to the previous analysis, I would first find the optimal values for the hyperparameters, following which I would test the performance of Q-learning on MountainCar.

4.3.1 Determining Learning Rate and Discount Factor:

Similar to Section 4.1.1 the learning rate and discount factor have been varied between 0.1 and 1.0. The algorithm has been run for 5000 episodes, with every episode running for 1000 episodes. I have also performed an epsilon decay strategy to ensure that the agent explores the system before learning. The idea of epsilon decay has been referred from [9] .

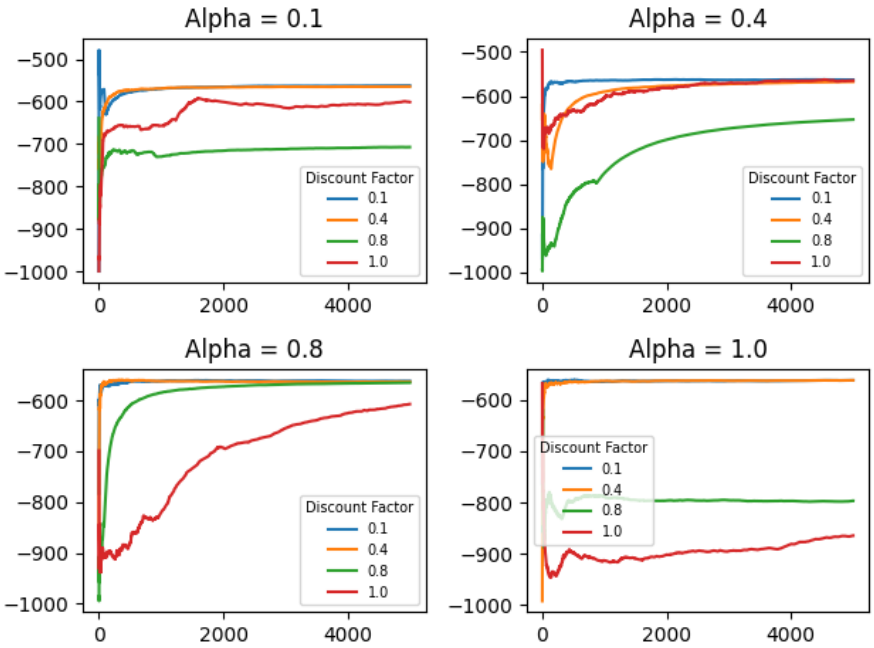


Figure 9: Q-learning performance against different learning rate and discount factor values

To ensure that I choose the optimal hyperparameter values, I have put an additional condition for termination. I have only terminated the training of the car, reaching the terminal position more than five times. This ensures that only those hyperparameters are selected to learn the optimal policy. The results can be found in Fig 9

From Fig 9 it is visible that my implementation of Q-learning was not able to solve this problem in 5000 episodes. I am not sure why the algorithm could not solve this problem as SARSA was able to solve it with similar settings. However, I have chosen the best hyperparameter values among the available choices to perform analysis. I believe the highest reward is obtained when the learning rate is 0.4. In that setting, the discount factors of 0.1, 0.4 and 1.0 give similar rewards. Hence I have randomly chosen the value of 0.4 as my discount factor. Thus the learning rate and discount factor both are set to 0.4. In the following section, I will test the performance of the Q-learning model on the MountainCar problem.

4.3.2 Performance of Q-learning:

In this section, I will test the performance of Q-learning on the MountainCar problem. The algorithm is run for 10,000 episodes, where each episode is run for 1000 timesteps. The algorithm is terminated when the car reaches the goal five times. The results of the performance are as shown in Fig 10. As visible in Fig 10 and as mentioned previously, the Q-learning model did not reach the goal in 10,000 episodes. If the Q-learning model had learnt properly, the algorithm would have terminated with a positive reward well before 10,000 episodes were completed. However, that is not the case as visible in Fig 10. Hence i believe my Q-learning implementation was **not** able to solve the MountainCar problem.

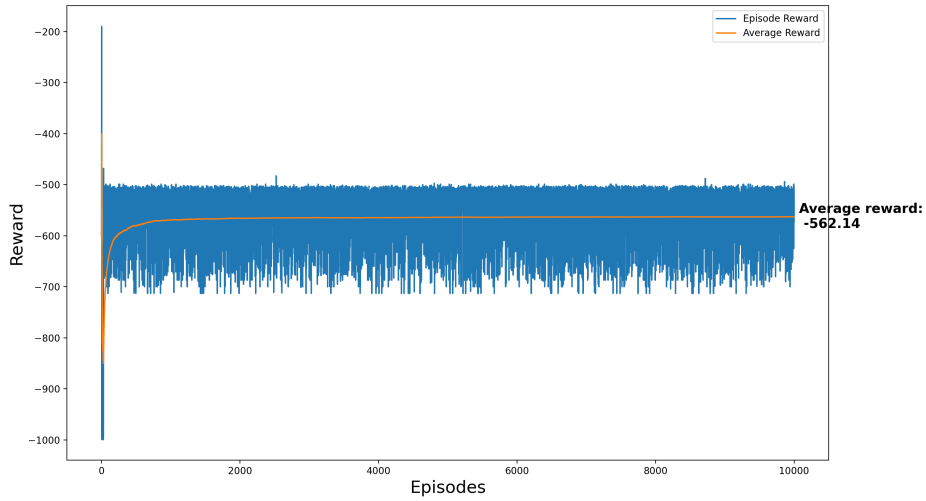


Figure 10: Q-learning with learning rate = 0.4 and discount factor = 0.4

5 Task 3:

5.1 On-policy and Off-policy:

In on-policy methods try to improve the same policy which is used by the agent to navigate the environment[15]. Similarly off-policy methods use two different policies. One policy is used by the agent to navigate through the environment while the second policy is the policy which actually learns from the environment.

SARSA is an example of on-policy model and Q-learning is an example of off-policy model. The reason SARSA is called as an on-policy model is because SARSA updates its current state Q-values by considering the **next action according to its current policy** [10]. This means that SARSA assumes that to move to the next state we have to follow the current policy and take the best action according to the policy[10]. Hence it is an on-policy model.

Q-learning is called an off-policy model because it follows 2 different policies. In Q-learning the agent is asked to take the next action greedily.[10] The agent takes the next action by not following the actual policy but by choosing the next action that has highest Q-value. However, when the update for the state-action value pairs happen, the updates happen according to the original policy and not the greedy policy. This means that in Q-learning, the agent follows a greedy policy while the update happens on the original policy[10]. Since two policies are followed in Q-learning, it is an off-policy method.

5.2 Model-based and Model-free Reinforcement Learning:

In model-based RL we need to have a model of the environment so that we can train the agent[15]. This means that in model-based RL the environment that we have modelled has an effect on the predictions/rewards of the agent [11]. Thus if we get the model wrong then our agent would never be able to successfully navigate through the environment. Dynamic Programming(DP) is an example of model-based RL[11]. In DP we need the transition probabilities among different states so that the agent can decide which state to choose that will maximise its reward. These transition probabilities are in a way a model of the environment[11]

In model-free RL, the agent has no predetermined environment model[15]. In fact the agent is initialised with some random values(also called as Q-values) and the agent learns to navigate through the environment on its own, by gaining experience over time[11]. Q-learning and SARSA are an example of model-free RL. These algorithms are initialised with random Q-values and the agent is run through the environment multiple times. As the agent gains more **experience** the Q-values are updated and eventually these algorithms converge to the optimal policy. Thus model-free methods learn from experience.

5.3 Policy-based and Value-based RL:

In Policy-based methods the agent chooses a random policy at the start. The initial policy does not have to be optimal. As the agent navigates throught the environment, for every action the agent takes a gradient is generated and we follow a gradient ascent strategy to maximise the expected reward. Thus using the gradient ascent strategy we try to find the optimal policy[12]. In policy-based methods, the agent directly learns the policy and the actions are sampled from the policy directly[12]. This ensures that there is some stochasticity in the policy based methods. REINFORCE is an example of policy-based methods.

In REINFORCE, we select a random policy and take N steps using that policy. Once the steps are completed we calculate the discounted reward and using backpropagation we odify the action probabilities for the policy. We can then perform gradient ascent to maximise the expected reward. This procedure followed by REINFORCE is very similar to the procedure mentioned above. Hence REINFORCE is a policy-based RL algorithm

In value based methods we learn the state-action value pairs for an environment. Using these state-action pairs we can determine the optimal policy. However, in these methods our aim is not to learn the policy directly, but to learn the state-action values and then find the optimal policy from those values. Since we do not sample actions from a probability distribution, value-based methods are deterministic by nature. Q-learning and SARSA are examples of value based methods.

5.4 Planning and Learning:

Planning as the name suggests involves pre-deciding agent's actions before the agent navigates through the environment. For example in model-based methods we tend to determine the model of the environment and in that process we "plan" how the agent should navigate through the environment [15]. Thus pre-deciding how the agent should navigate through the environment is called planning[15]. Dynamic Programming(DP) is an example of planning before the agent navigates through the system. In DP the transition probabilities are determined before the agent starts its navigation[15]. Thus determining the transition probabilities means determining the agent's actions before the agent starts navigating through the environment. This is known as planning.

Learning as the name suggests does not involve any planning. This means that we do not tell the agent any additional information and we let the agent explore the environment. The agent learns through experience and decides the optimal policy[15]. SARSA and Q-learning are examples of learning mechanism. These algorithms update the Q-values as they navigate through the environment and eventually they learn the optimal policy.These algorithms have no pre-assumption about the environment and all the experience is gained after navigating through the environment.

5.5 Offline and Online RL:

In RL the agent tries to navigate through the environment and during that process it gains experience. Using this experience the policy is modified and when repeated a large number of times we get an optimal policy. There are 2 categories through which an agent can gain this experience.

The first category is online learning, where the agent is made to run in an actual real world environment. Thus using the real world experience the agent tries to learn the optimal policy[14]. Q-learning and SARSA could be given as examples of online RL as these models are generally trained on actual environment.Even though, online RL is an ideal way to train an agent, this method is very time consuming, expensive and sometimes even dangerous [14]. Hence, researchers are trying to explore another alternative which is described below.

The second method is an offline method. In this method the agent tries to gain experience from data which has been collected and stored. This setting is very similar to a supervised learning method where a model tries to learn from a pre-collected and stored dataset[14]. This method is much more viable because it is less time consuming and less expensive. However, this method suffers from problems of its own. For example, this method is known ti suffer from the problem of distributional shifts[14] etc. Recently an algorithm called "Conservative Q-learning" was intrduced in the field of offline-RL [13]. This model tends to tackle the problem of distributional shift by conservatively placing the importance on future unseen values.

References

[1] URL: https://gym.openai.com/envs/CartPole-v0/.

[2] URL: https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py.

[3] URL: https://github.com/philtabor/Youtube-Code-Repository/blob/master/ReinforcementLearning/Fundamentals/sarsa.py.

[4] URL: https://towardsdatascience.com/how-to-beat-the-cartpole-game-in-5-lines-5ab4e738c93f.

[5] URL: https://shiva-verma.medium.com/solving-reinforcement-learning-classic-control-problems-openaigym-1b50413265dd.

[6] URL: https://github.com/JackFurby/CartPole-v0/blob/master/cartPole.py.

[7] URL: https://gym.openai.com/envs/MountainCar-v0/.

[8] URL: https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py.

[9] URL: https://github.com/philtabor/Youtube-Code-Repository/blob/master/ReinforcementLearning/Fundamentals/mountaincar.py.

[10] URL: https://stats.stackexchange.com/questions/184657/what-is-the-difference-between-off-policy-and-on-policy-learning.

[11] URL: https://stats.stackexchange.com/questions/184657/what-is-the-difference-between-off-policy-and-on-policy-learning.

[12] URL: https://www.youtube.com/watch?v=cQfOQcpYRzE.

[13] Aviral Kumar et al. “Conservative q-learning for offline reinforcement learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[14] Sergey Levine et al. “Offline reinforcement learning: Tutorial, review, and perspectives on open problems”. In: *arXiv preprint arXiv:2005.01643* (2020).

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.