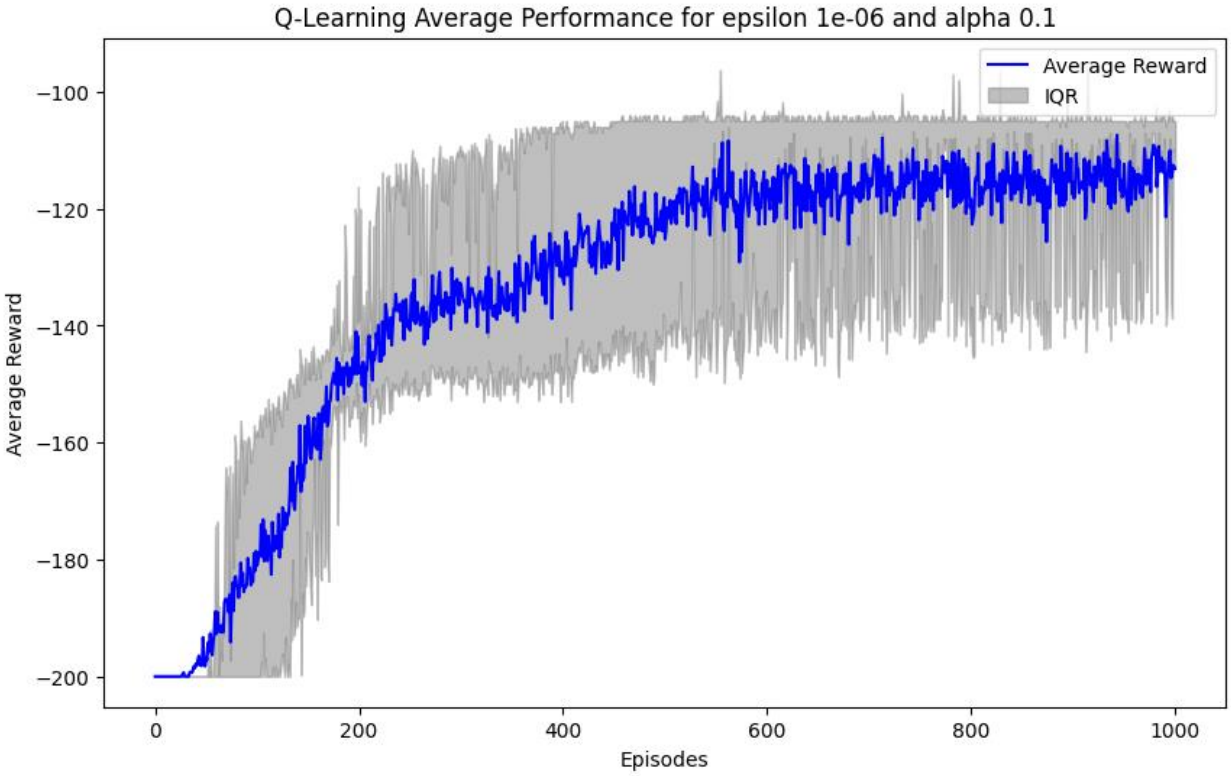
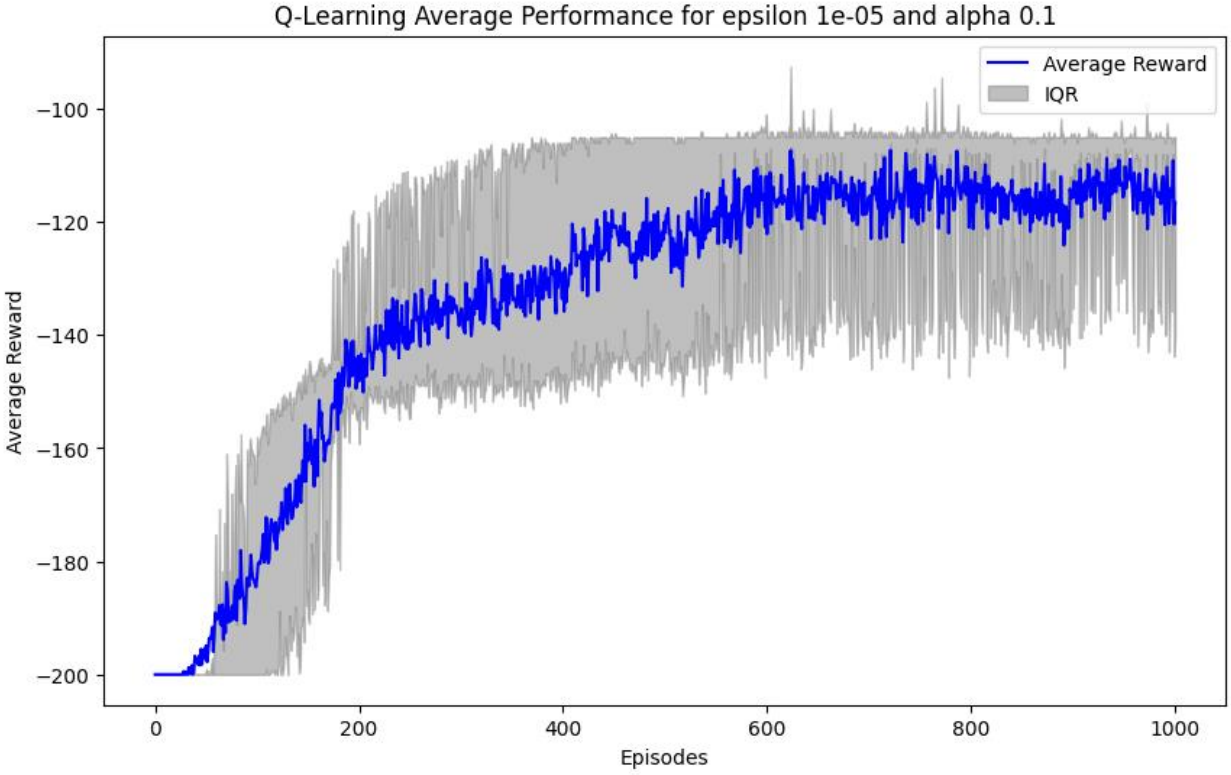
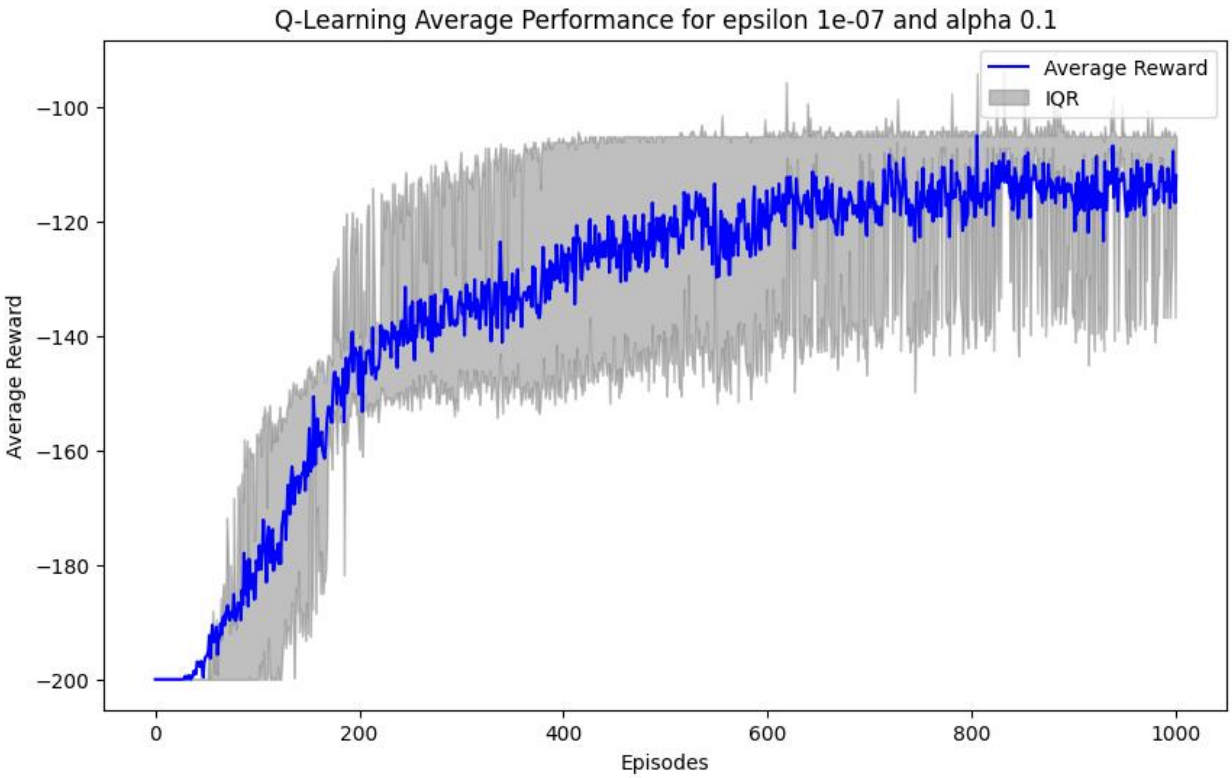
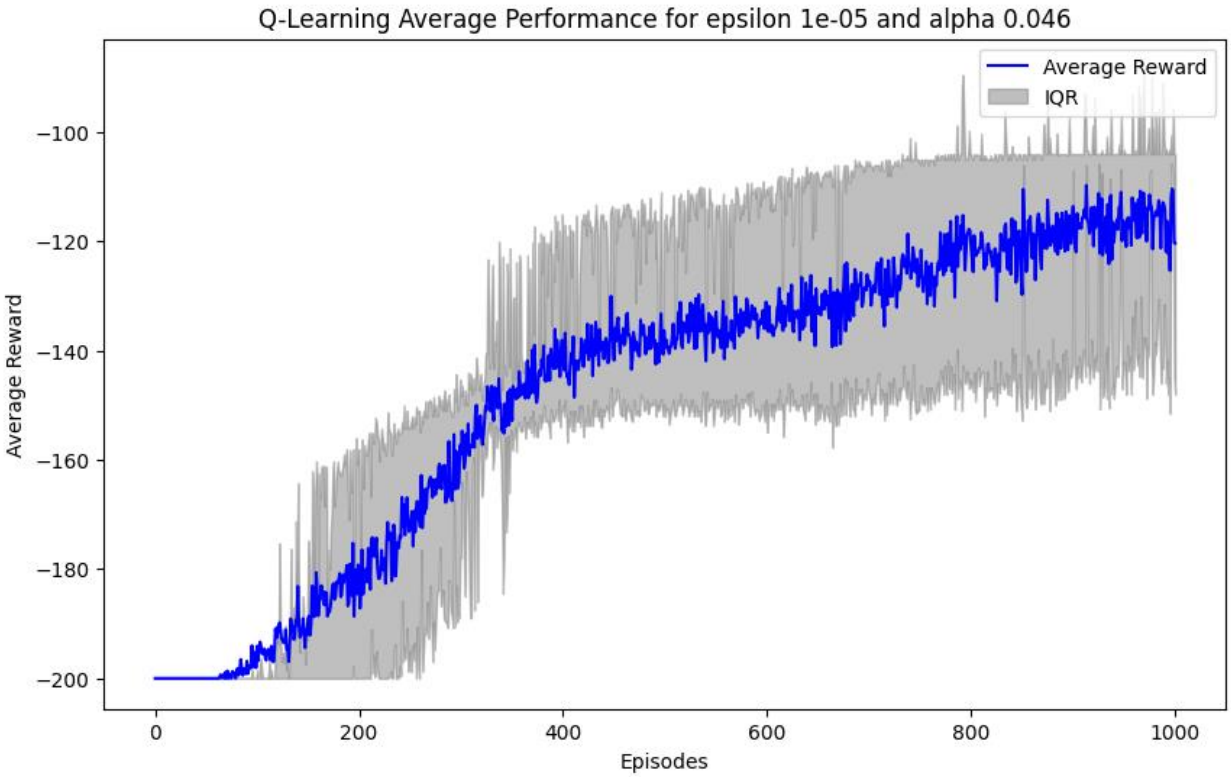
Q1Report and Graphs

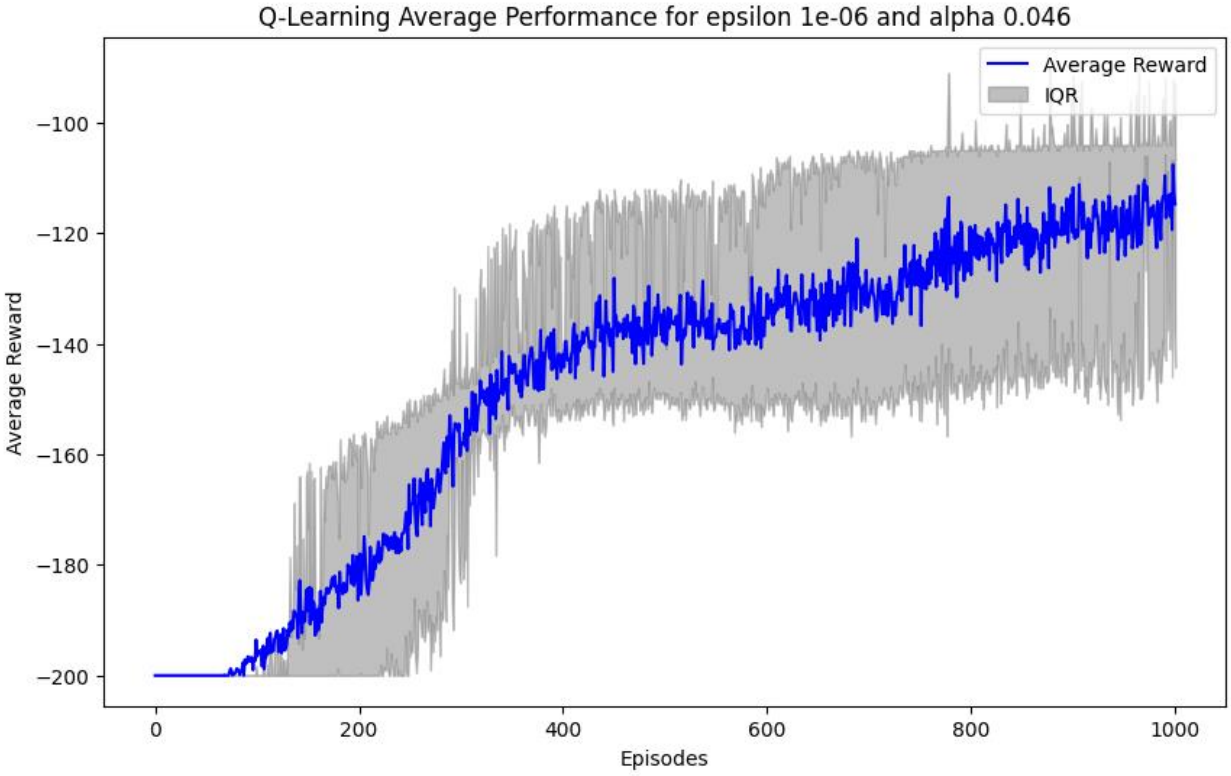
Mountain Car Domain Q-Learning

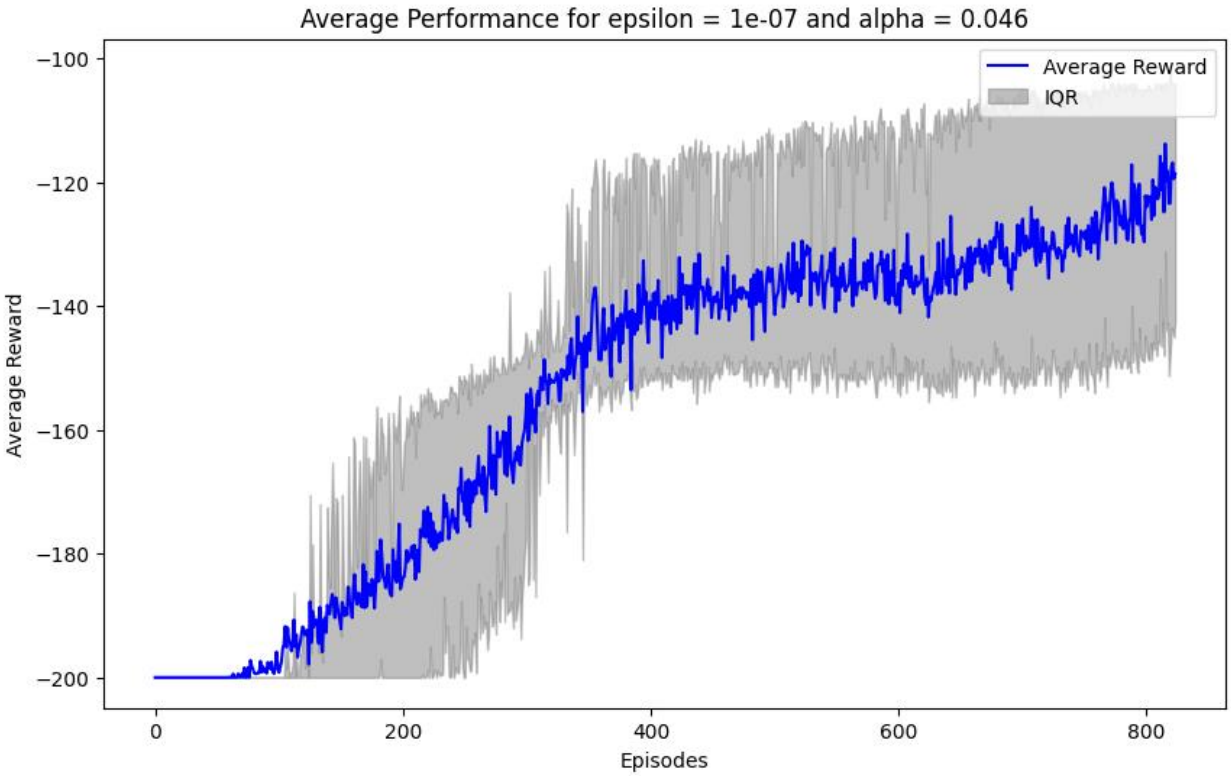


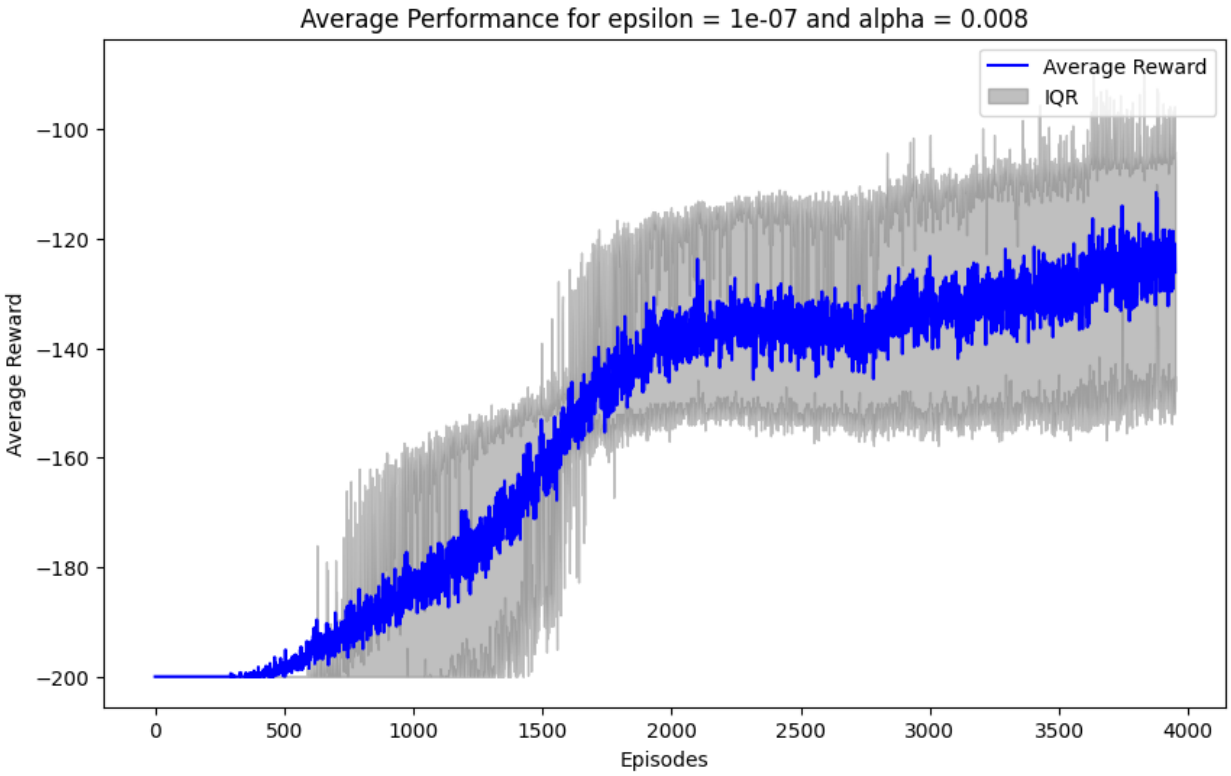


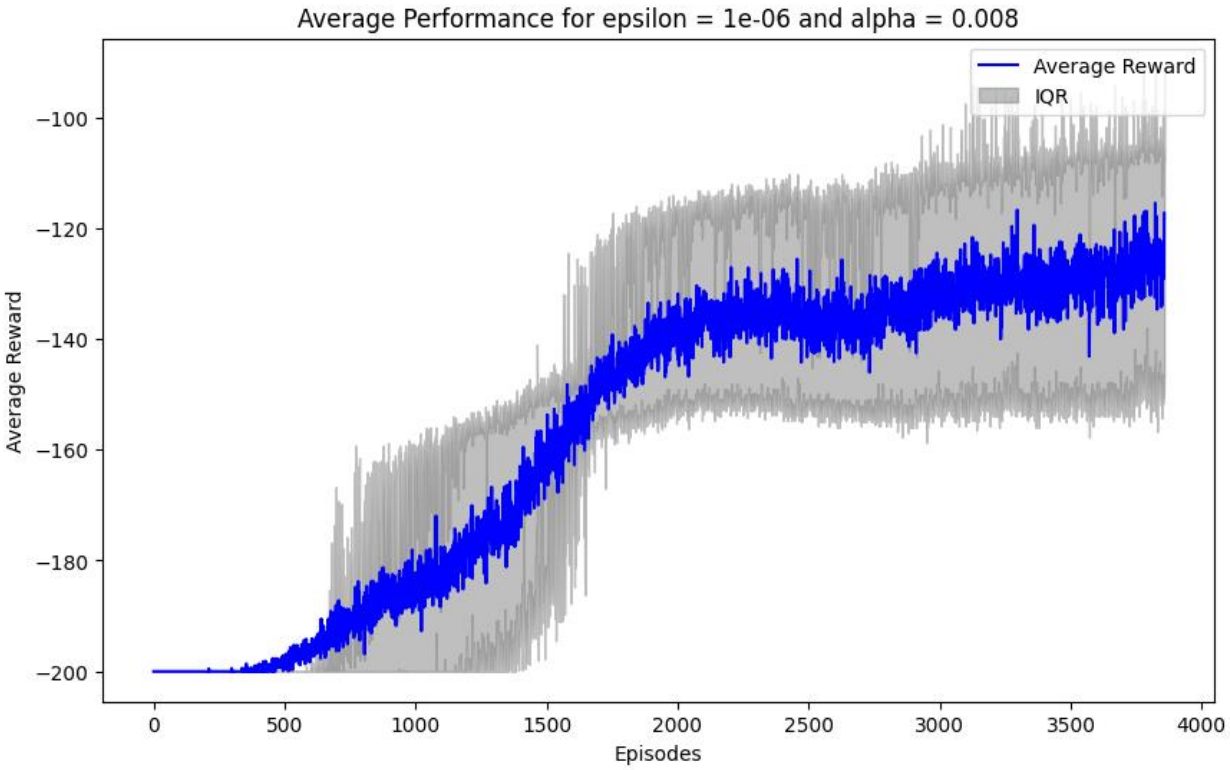

Q-Learning Average Performance for epsilon 1e-07 and alpha 0.1
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
Episodes

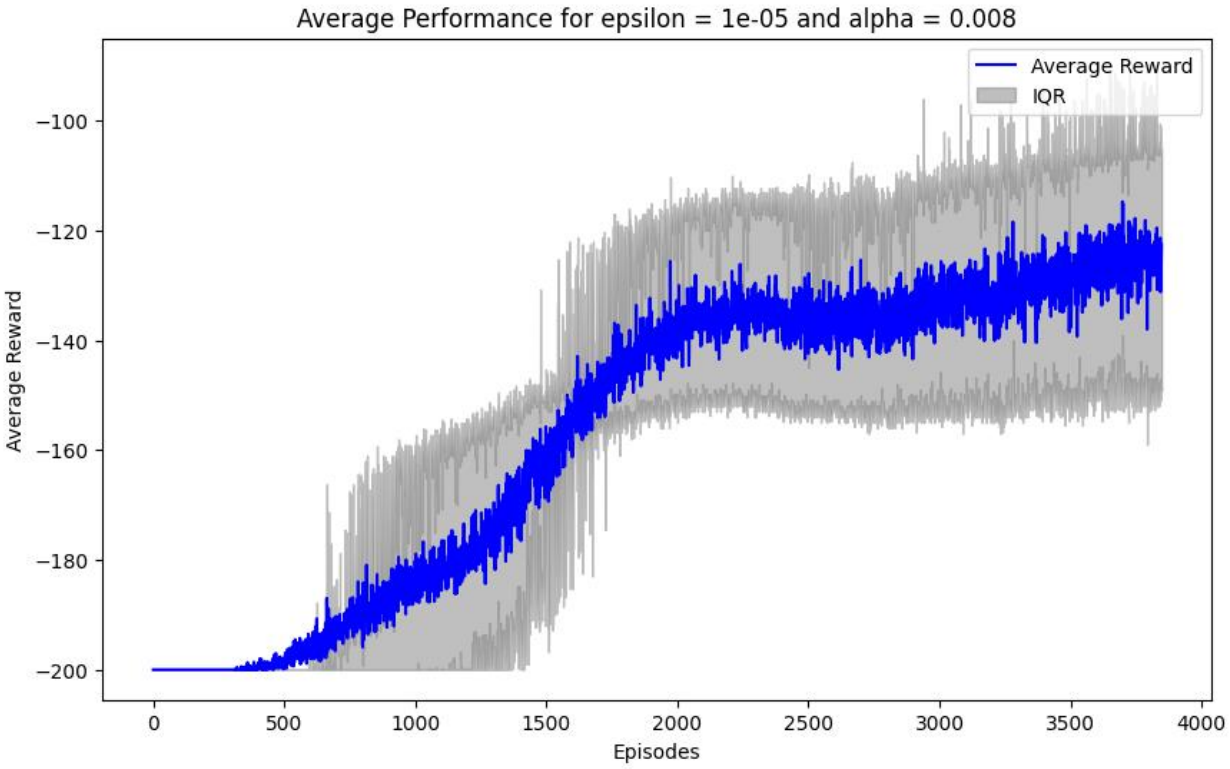

Q-Learning Average Performance for epsilon 1e-05 and alpha 0.046
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
Episodes


Q-Learning Average Performance for epsilon 1e-06 and alpha 0.046
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
Episodes


Average Performance for epsilon = 1e-07 and alpha = 0.046
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
Episodes


Average Performance for epsilon = 1e-07 and alpha = 0.008
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
500
1000
1500
2000
2500
3000
3500
4000
Episodes





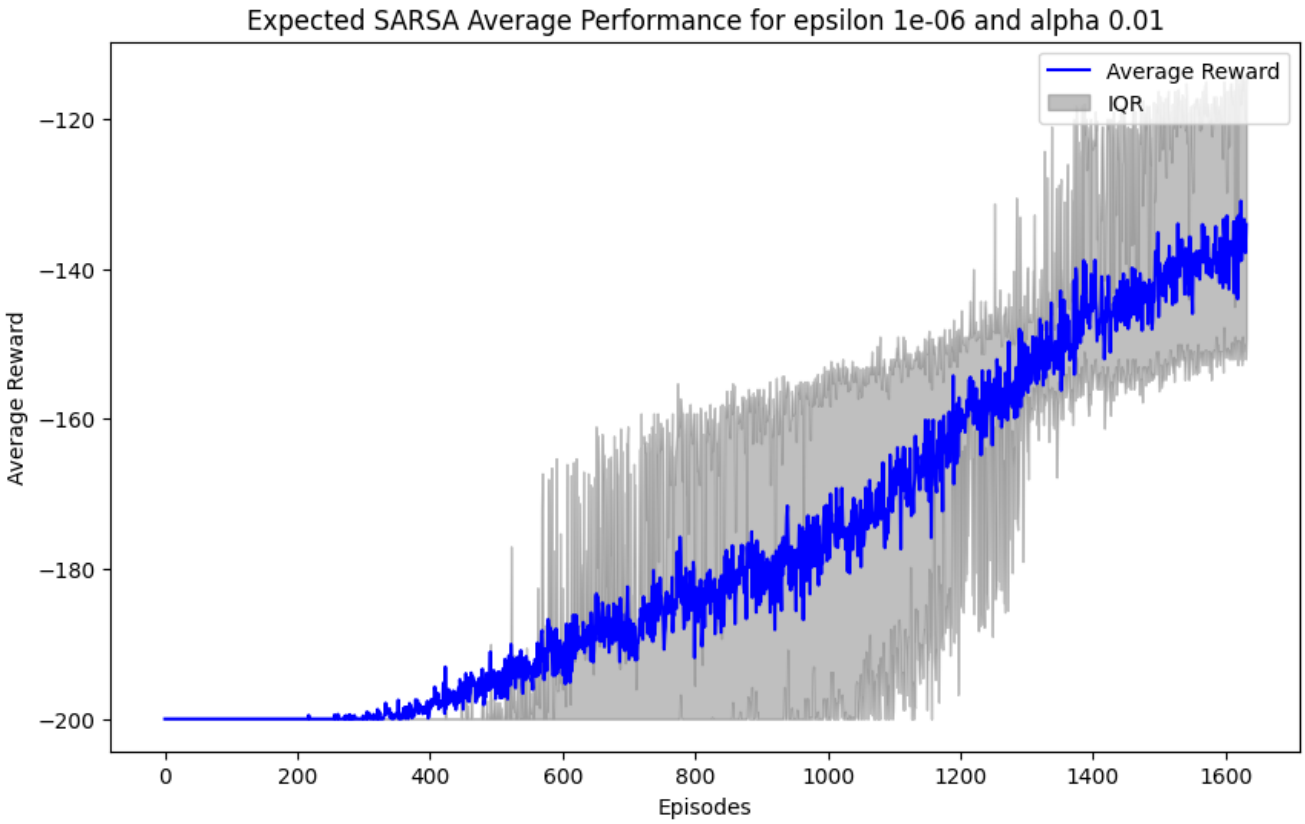
We remark a difference in the shape of the curve. For Q-Learning it resembles a sigmoid function with a rapid improvement and then a plateau whereas Expected SARSA takes more time to improve

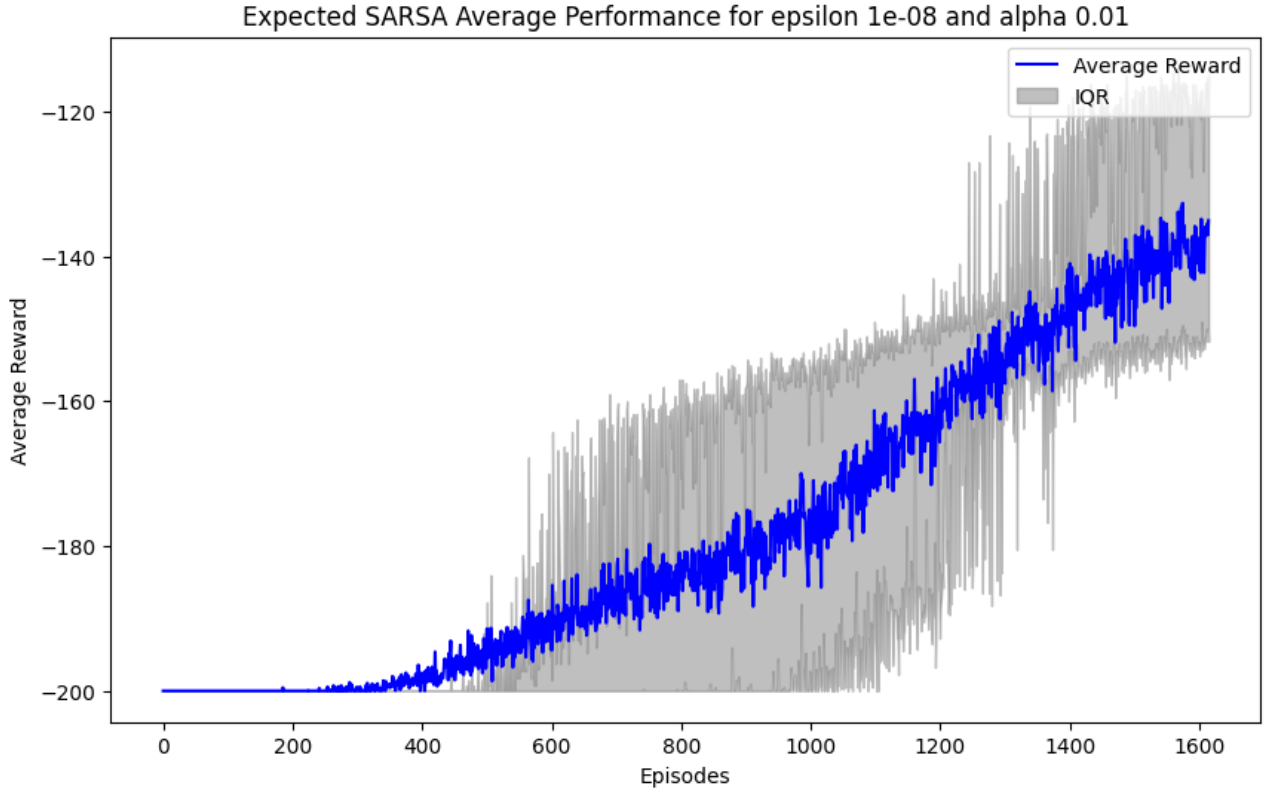
and doesn't plateau before the end of the maximum number of episodes allocated. It likely plateaus late on somewhere between -120 and -140 but would need more episodes. This is difficult to do with 50 runs since the graphs we have took us approximately 2 days to get.

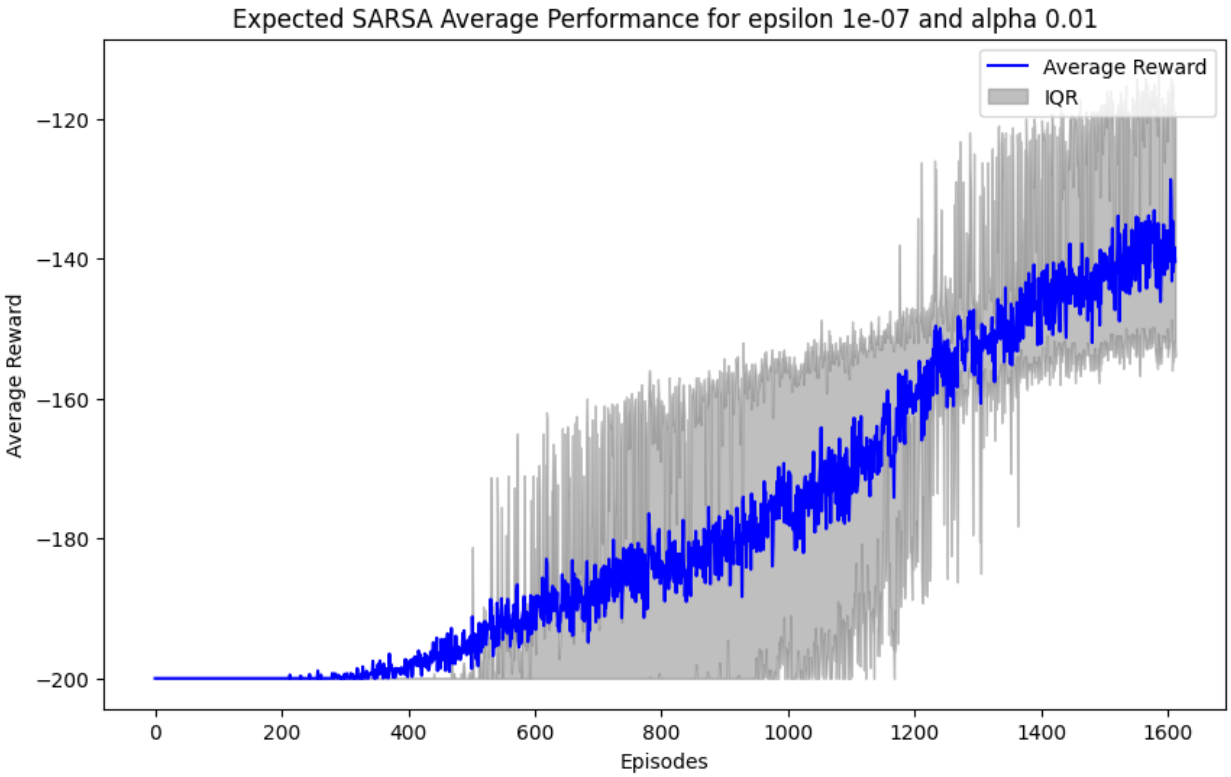
We remark that both algorithms are highly sensible to alpha which controls the speed of learning. This was expected. We see that the higher the alpha, the faster the algorithm converges. That being said, we do not necessarily remark that with a higher alpha the algorithm converges to a lower reward. Nevertheless, when trying high alphas like 0.5, we remark weird behaviour such as the agent being stuck at a reward of -200.

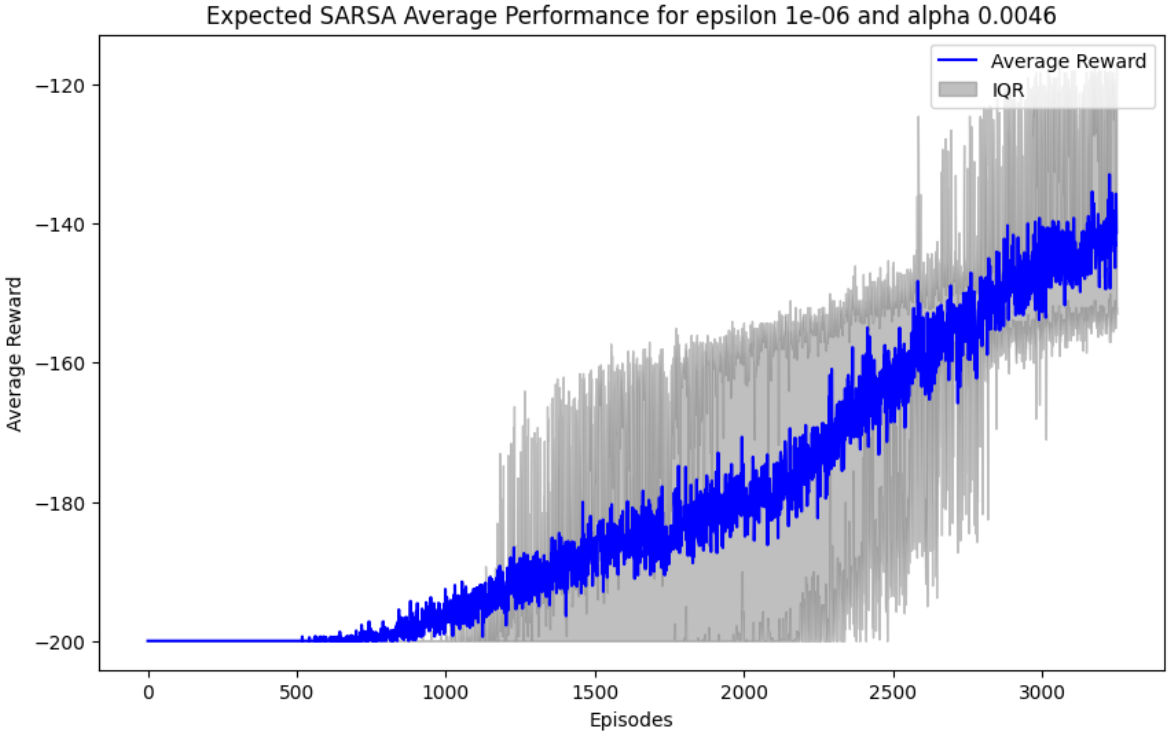
After trying several epsilons, we remark that epsilon-greedy is not the best strategy for both algorithm and we should instead choose argmax. This is the reason for the very low epsilons.

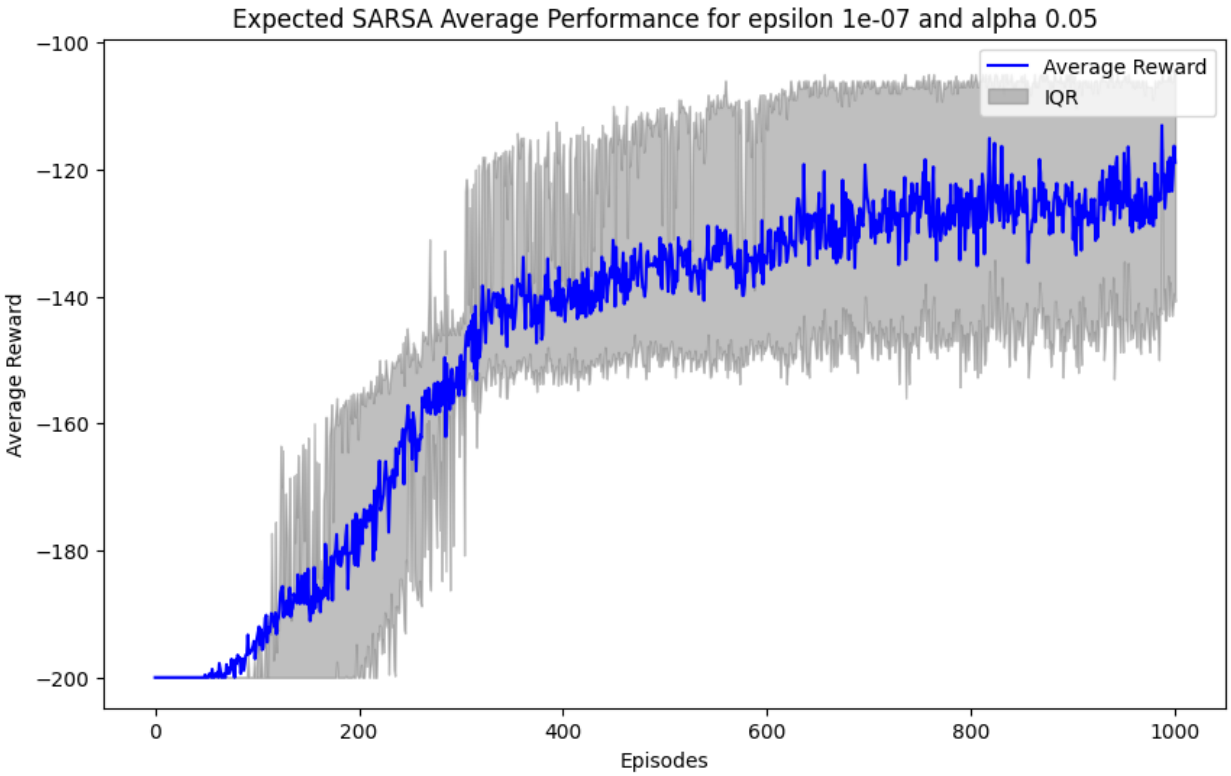
Below are the results for Expected SARSA in the Mountain car domain.

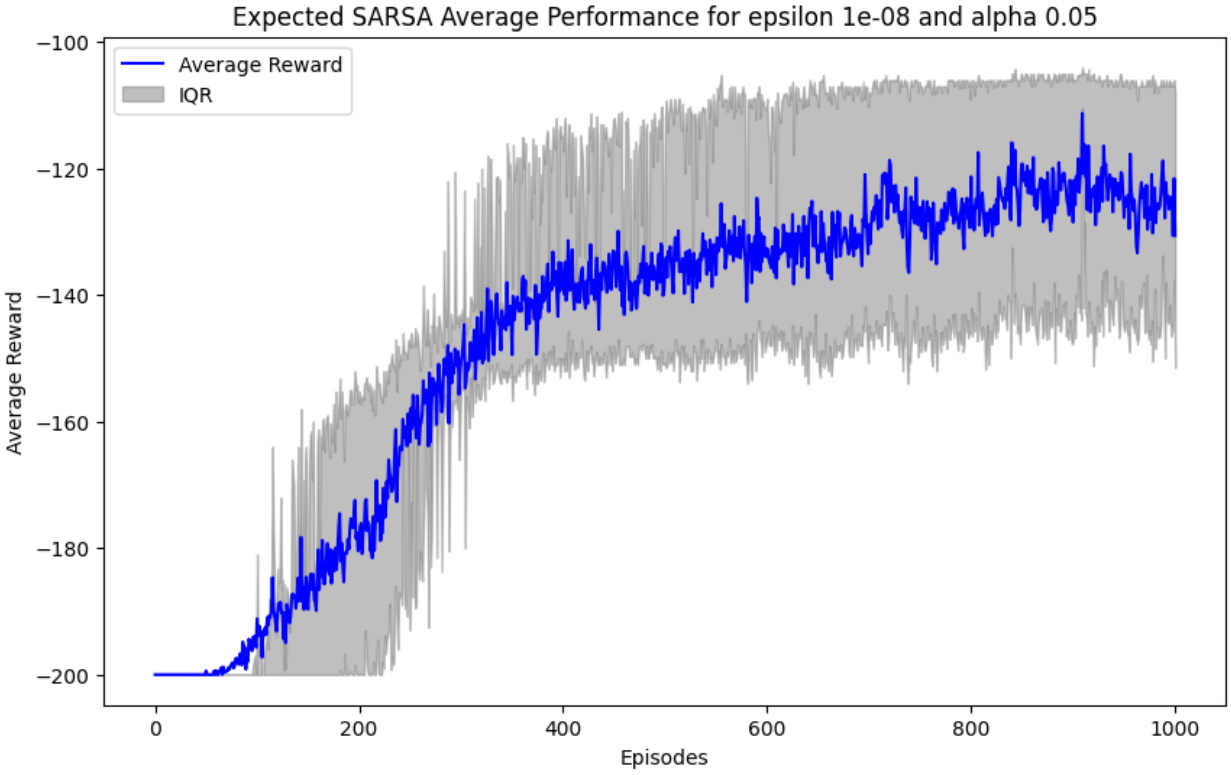

Expected SARSA Average Performance for epsilon 1e-06 and alpha 0.01
Average Reward
IQR
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
1200
1400
1600
Episodes

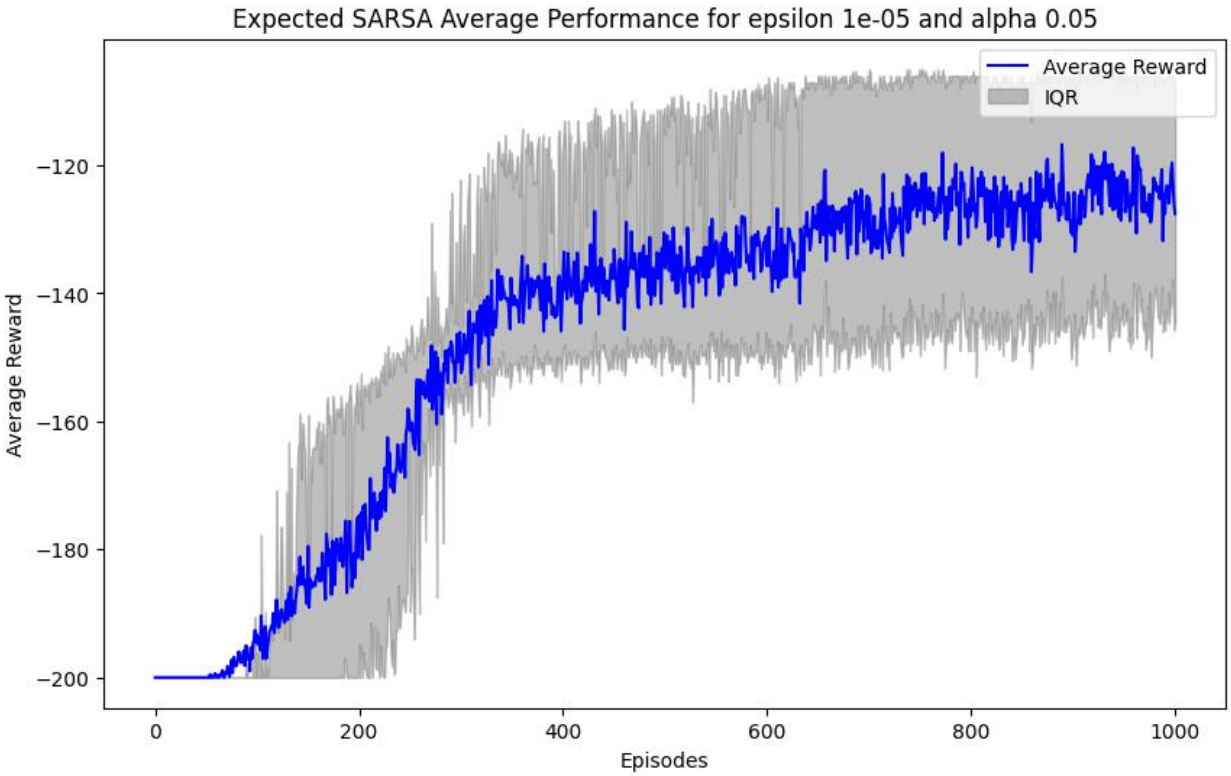

Expected SARSA Average Performance for epsilon 1e-08 and alpha 0.01
Average Reward
IQR
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
1200
1400
1600
Episodes


Expected SARSA Average Performance for epsilon 1e-07 and alpha 0.01
Average Reward
IQR
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
1200
1400
1600
Episodes


Expected SARSA Average Performance for epsilon 1e-06 and alpha 0.0046
Average Reward
IQR
−120
−140
−160
−180
−200
Average Reward
0
500
1000
1500
2000
2500
3000
Episodes


Expected SARSA Average Performance for epsilon 1e-07 and alpha 0.05
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
Episodes


Expected SARSA Average Performance for epsilon 1e-08 and alpha 0.05
Average Reward
IQR
−100
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
Episodes


Expected SARSA Average Performance for epsilon 1e-05 and alpha 0.05
Average Reward
IQR
−120
−140
−160
−180
−200
Average Reward
0
200
400
600
800
1000
Episodes

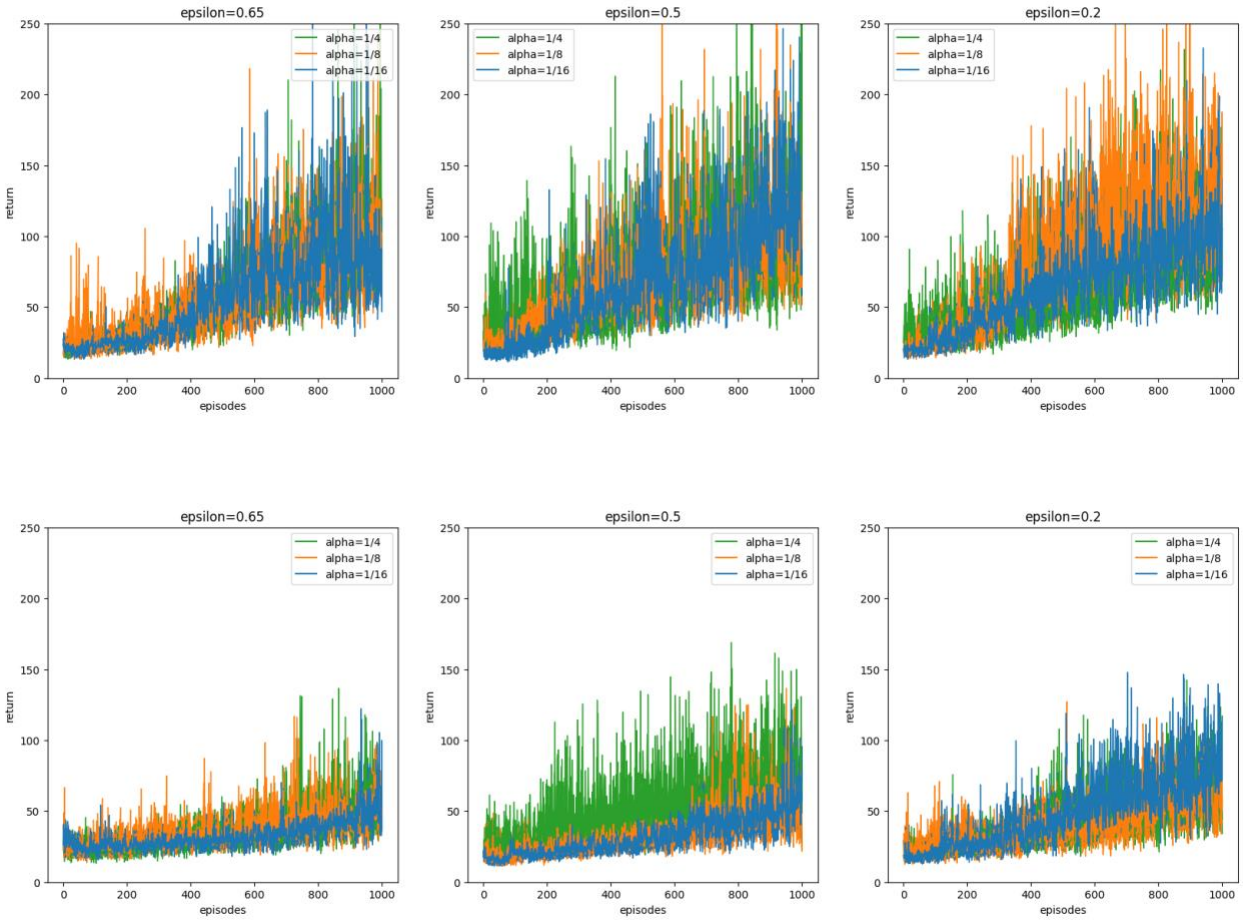
CartPole Domain

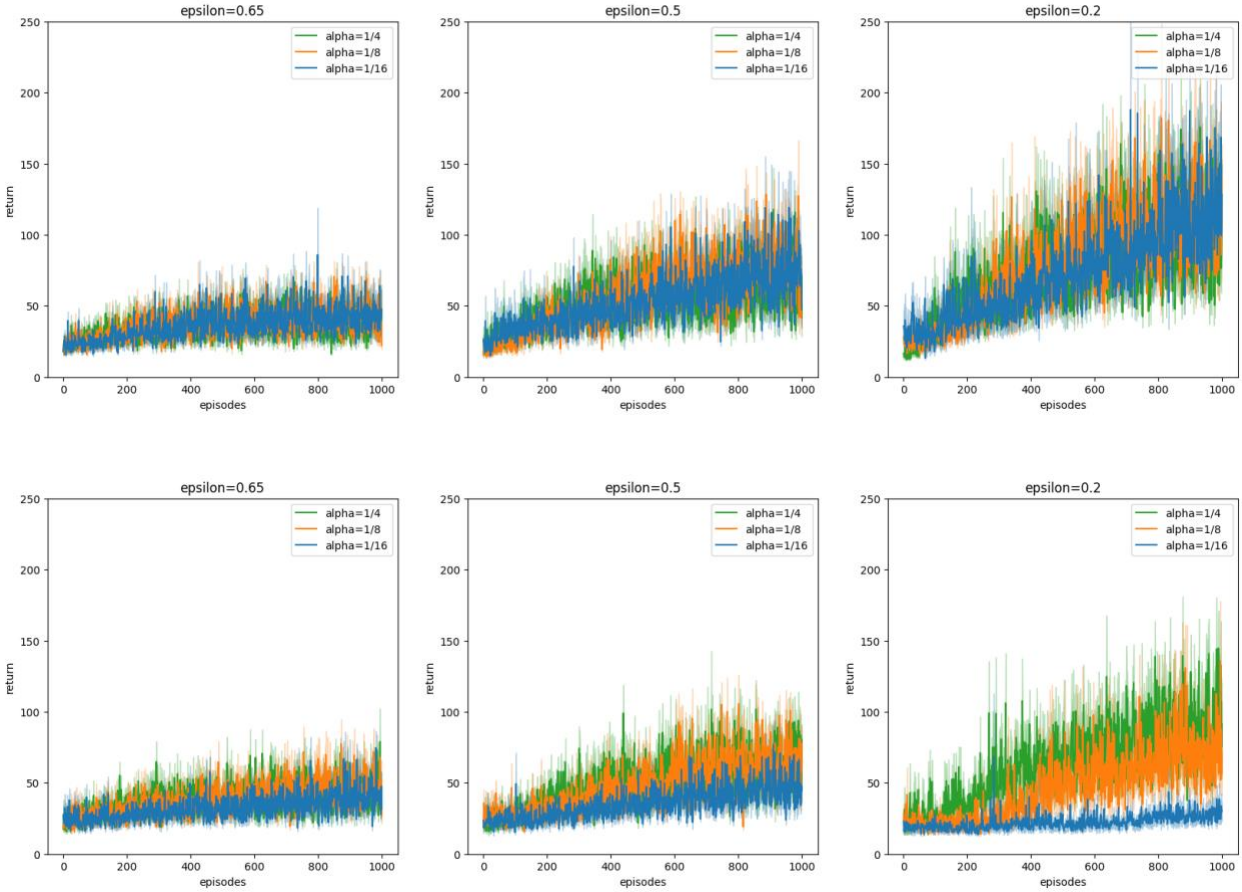
All the below is with `epsilon_minimum = 0.2`

Whenever we lower the lower bound of epsilon (epsilon_minimum) we get results that are far worse than the ones below.

Here are two sets of 6 graphs. The first and third line of graphs represent the Q-learning algorithm while the second and fourth lines represent the expected-SARSA algorithm. We remark that for both sets and also in both domains q-Learning performs better than expected SARSA. Here, the difference is quite noticeable. This is potentially because of the inherent nature of the CartPole domain or because of our choices in the function approximation.

The results in the first set of graphs (first two lines below) are better because we decided to decay the epsilon as we weren't satisfied with the results in the second sets. Decay allows the agent to explore at the beginning and slowly but surely decide on a more greedy policy which is better overall.




epsilon=0.65
epsilon=0.5
epsilon=0.2
alpha=1/4
alpha=1/8
alpha=1/16
return
episodes
0
50
100
150
200
250
0
200
400
600
800
1000

epsilon=0.65
epsilon=0.5
epsilon=0.2
alpha=1/4
alpha=1/8
alpha=1/16
return
episodes
0
50
100
150
200
250
0
200
400
600
800
1000

Comp579 Assignment 3 - Q3

Herbie He, Mohamed Tliouant

1. Implementation

1.1. REINFORCE

We choosed to use a 1-hidden layer (with 128 units) Neural Net with ReLU activation as our policy. The gradient for the two sets of weights are computed manually.

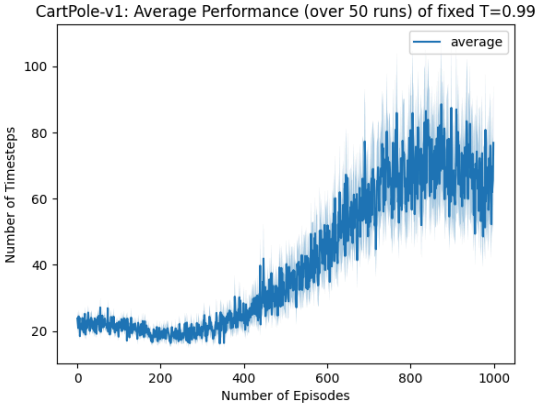
1.2. Actor Critic

2. CartPole

2.1. Reinforce

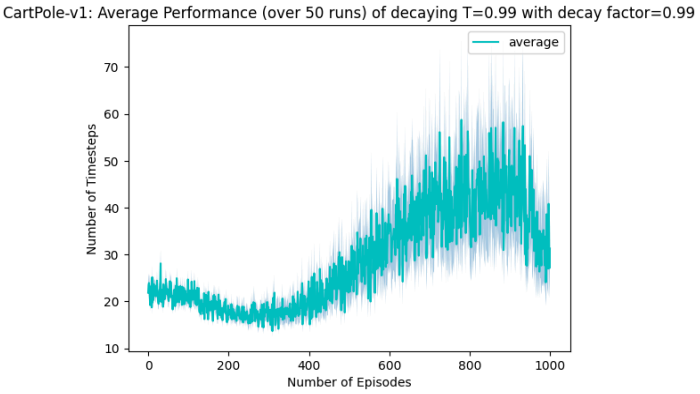
Fixed -T

After tuning, we decided to use T=0.99, $\gamma = 0.5$, $\alpha = 4e\text{-}4$, which gives us the following result



Decaying-T

With all other parameters unchanged, except for adding a decay factor =0.99 for T, we get the following result.



Observation

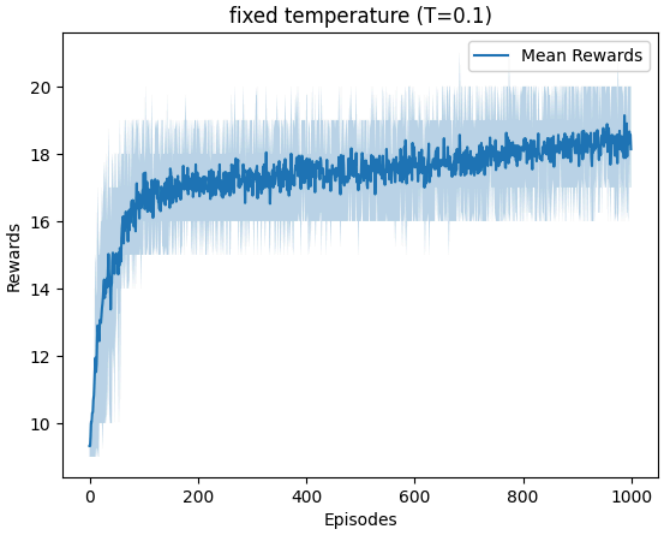
- Comparing fixed T and varying T with REINFORCE, we notice they both exhibit a parabola-like pattern where the performance peaks at 900 episodes. Using a fixed T gives us 2x better results than using a decaying T.
- In Boltzman's equation, small T magnifies the preference for the greedy action. Theoretically, it seems better to decay T after the agent has learned more about the environmen and the pros and cons of each actions at each state after acting quite stocastically. However, our observation contradicts that. One possible explanation is that, in continuous state space, its very unlikely for the agent to explore all states and all actions and constructs a comprehensive "guide" of

what to do in each case. When in a novel state, some times its better to act with some randomness than trying to match the state to some previous state and apply some previous action that may not be optimal under current setting.

2.2. Actor Critic

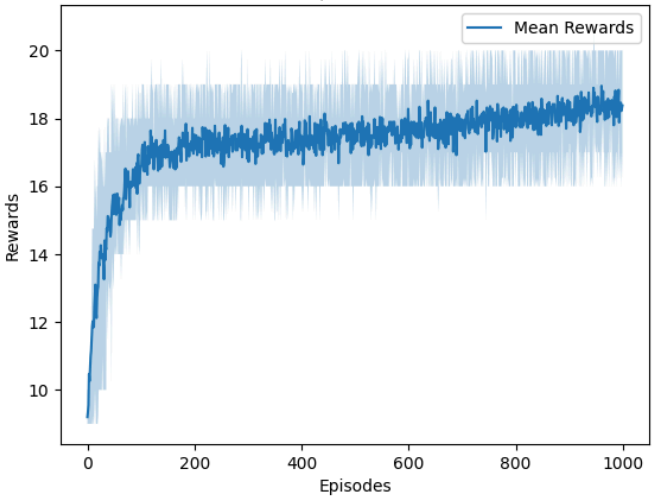
Fixed T

After tuning, we decided to use T=0.1, α = 5e-5 (learning rate for critic), and $\beta = 0.5$ (learning rate for actor) which gives us the following result



Decaying-T

With all other parameters unchanged, except for adding a decay factor =0.9 for T, we get the following result.



Observation

Fixed-T and Decaying-T Actor Critic exhibits almost identical bheaviour where the performance rises rapidly initially and enters a steady phase for the rest of the episodes. However, it would be interesting to see how they differ in the long run.

2.3. Comparing Different Algorithms

Comparing REINFORCE and Actor Critic

- Actor Critic learns much faster than REINFORCE. This may be partly due to the Monte-Carlo style learning adopted by REINFORCE, where the weights are only updated and applied after the entire episode. So the agent makes progress on episode basis. With Actor Critic, we update

and apply our updated weights at everytime steps. So the agent make progress on timestep baiss

- REINFORCE achieves better performance. As we can observe, REINFORCE peaks at around 80 and 60 for fixed-T and decaying-T respectively while Actor Critic peaks at 20 for both configurations. However, its not apparent whether Actor Critic will outperform REINFORCE in the long run as the curve of Actor Critic is steadily increasing while REINFORCE's seems to fall after 1000 episodes

3. Mountain Car

The Mountain Car problem is significantly more difficult to tackle than Cartpole. Firstly, the agent needs to learn to build a long chain of consistent actions in order to succeed. For example, the agent need to frist accelerate to the left a dozen times, to gain momentum, then accelerate consistently to the right to swing up to the top. If the agent decides, at any timestep, to move in the opposite direction, which breaks the integrity of this chain of actions, the agent will fail and have to repeat the entire process again.

Secondly, the algorithm can't learn anything even for long runs (using many episodes) because it's nearly impossible for the agent to randomly move to the top and receive its first non-negative reward, and thus, improves the policy. So essentially, the lack of intermediate rewards/incentives is the biggest barrier. To overcome this, **we have designed a different reward mechanism**, inspired by this discussion, that incentivizes the agent when it makes some progress. This set of reward rules is shown to be very effective, and the agent is able to consistently reach the top of the hill after only 100 episodes (using REINFORCE) as supported by our data (averaged over 20 trials).

Reward Mechanism:

- The agent is given a reward every time it makes it past its previous max position.
  - The reward is proportional to the distance between its current position (max position) and the bottom of the valley.
  - The magnitude of the reward is parameterized by a hyper-parameter ***Positive Reward Factor (PRF)***
- The agent is given a reward every time it makes it past its previous min position.
  - The reward is proportional to the distance between its current position (max position) and the bottom of the valley.
  - The magnitude of the reward is parameterized by a hyper-parameter ***Negative Reward Factor (NRF)***
- The agent is unconditionally given a reward proportional to its current velocity (absolute value)
  - The magnitude of the reward is parameterized by a hyper-parameter ***speed factor (SF)***

The intuition is to incentivize the agent to move away from the bottom of the valley as much as possible and achieve as fast speed as possible. Besides changing the reward function, we also increased the truncation timestep to 1000, allowing the agent to have more time to reach the top.

This reward mechanism is applied to REINFROCE and Actor Critic

Data Collection

Since we've defined our reward mechanism, it is reasonable to measure the total rewards acquired per episode as an indication of how well the agent has learned over the episodes. Ideally, we want the total reward to increase over the episodes since it signals that the algorithm is making progress even though the agent might fail to reach the top.

We also collected the maximum position of the agent per episode. Ideally, we would also want the max position to increase over episodes as it indicates the agent is getting closer to the top.

At last, we collected the timesteps per each episode. Ideally, we want the timesteps to drop below 1000, which indicates that the agent is successfully reaching the top.

3.1. REINFORCE

Exploring the effect of PRF, NRF, and SF

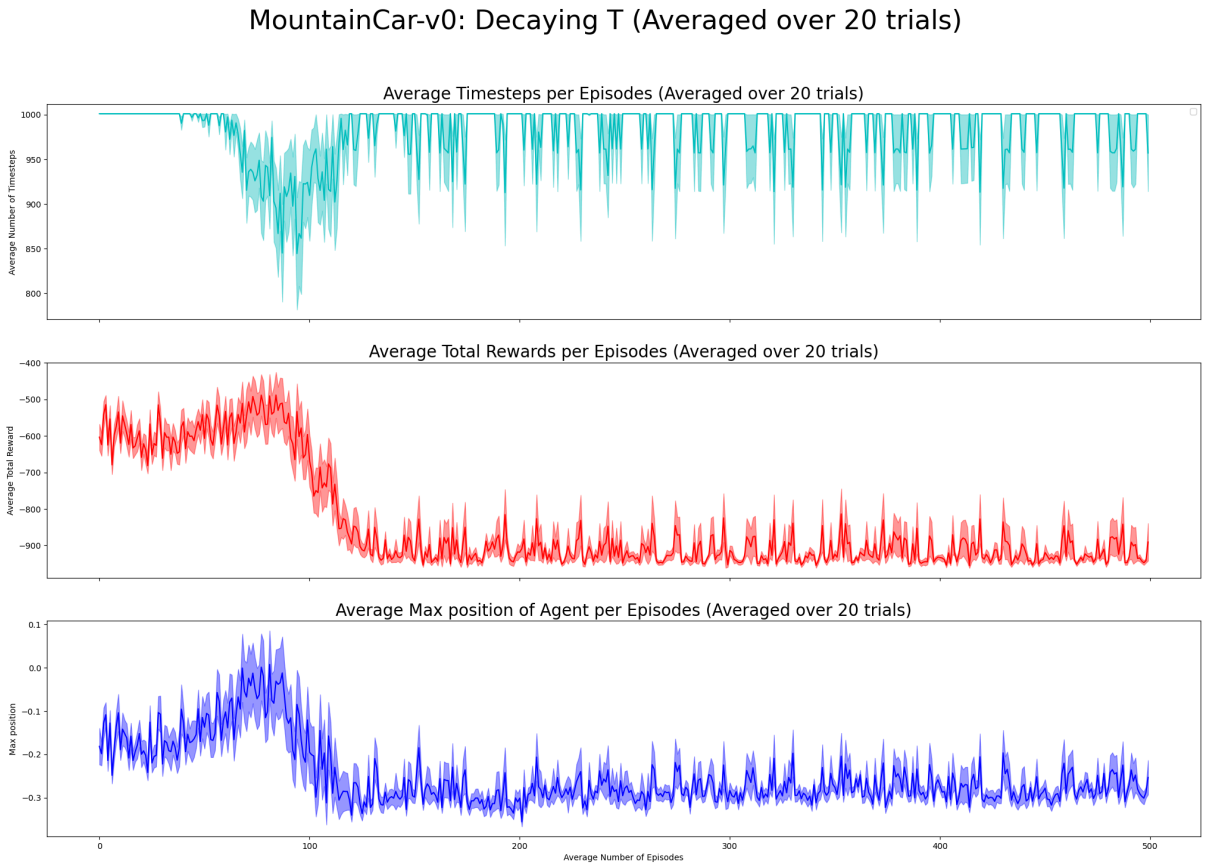
After testing with different combinations of PRF, NRF, and SF, we found that PRF=0.1, NRF=10, and SF=10 give surprisingly good results.

PRF=0.1, NRF=10, and varying SF. As we can observe, SF=10 (rewarding agent more for higher velocity) leads to a higher success rate with multiple episodes terminating at 200-time steps and significantly outperforms other SF values.

Decaying T

The following plots display the time steps, total rewards, and max position per episode, averaged over 20 trials with $\alpha = 1e-5$, $\gamma = 0.5$, $T = 0.1$, $PRF = 0.1$, $NRF = 10$, $SF = 10$. We've noticed from our past experiment that patterns began to emerge after 50 episodes. Therefore we chose to only run 500 episodes. Under our implementation, running 20 trials costs 4hrs, and running the full 50 trials costs 9 hrs, so we had to decrease the number of trials from 50 to 20 to save time (and our laptops). Nevertheless, we are able to acquire a very clear pattern.

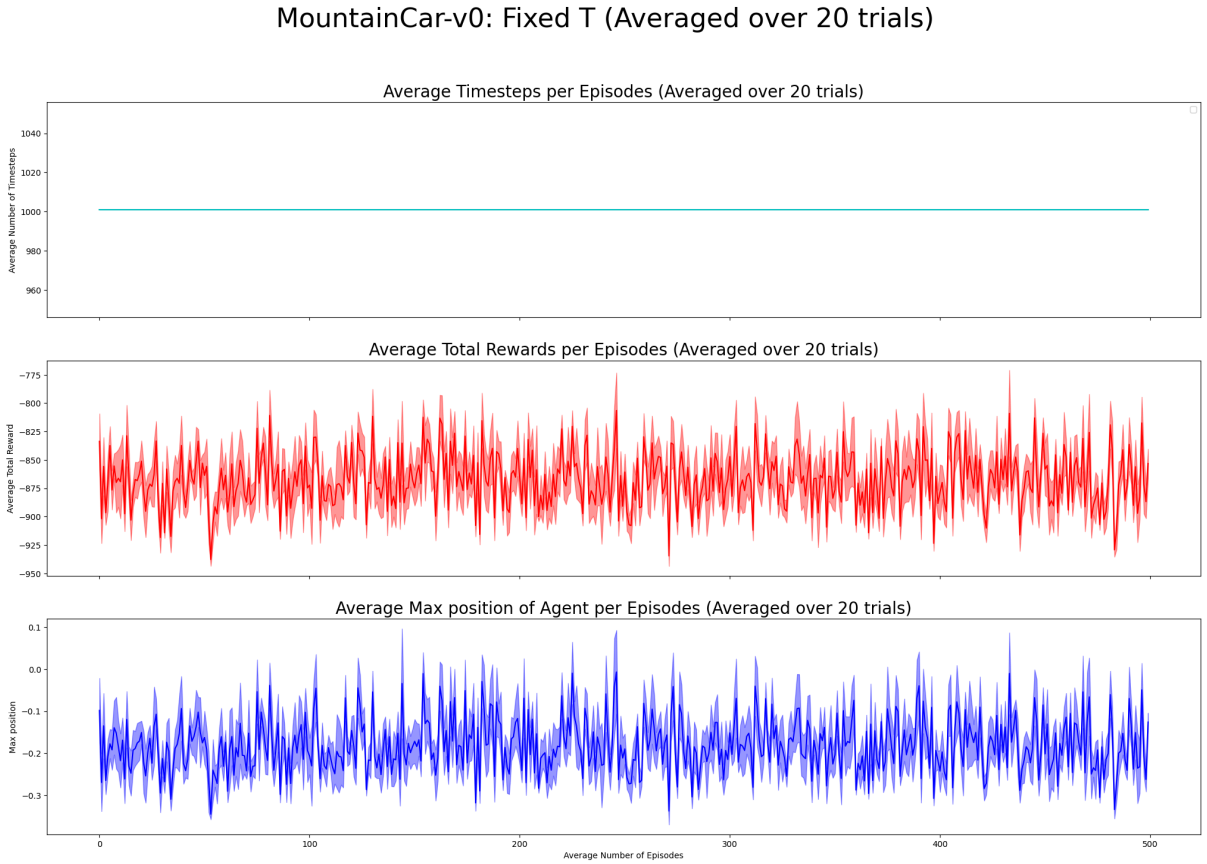
- The agent starts to consistently reach the top of the hill after 75 episodes
- The best performance of the agent is around 800 time steps (succeeding after 800 time steps)
- Even though the agent lost its consistency of succeeding after 120 episodes, its still able to succeed quite frequently, as apparent from the spikes in the first plot.



Fixed T

After tuning, we choosed to use $\alpha = 1e-5$, $\gamma = 0.5$, $T = 10$, $PRF = 0.1$, $NRF = 1$, $SF = 10$, which gives us the following result.

- The agent failed to make noticeable progression through the episodes. This is manifested by the 3rd subplots where we can see that the agent stays at the bottom of the valley for most episodes.
- The agent failed to succeed within 1000 timesteps for most episodes as we can see from 1st subplot that there is rarely any spikes.

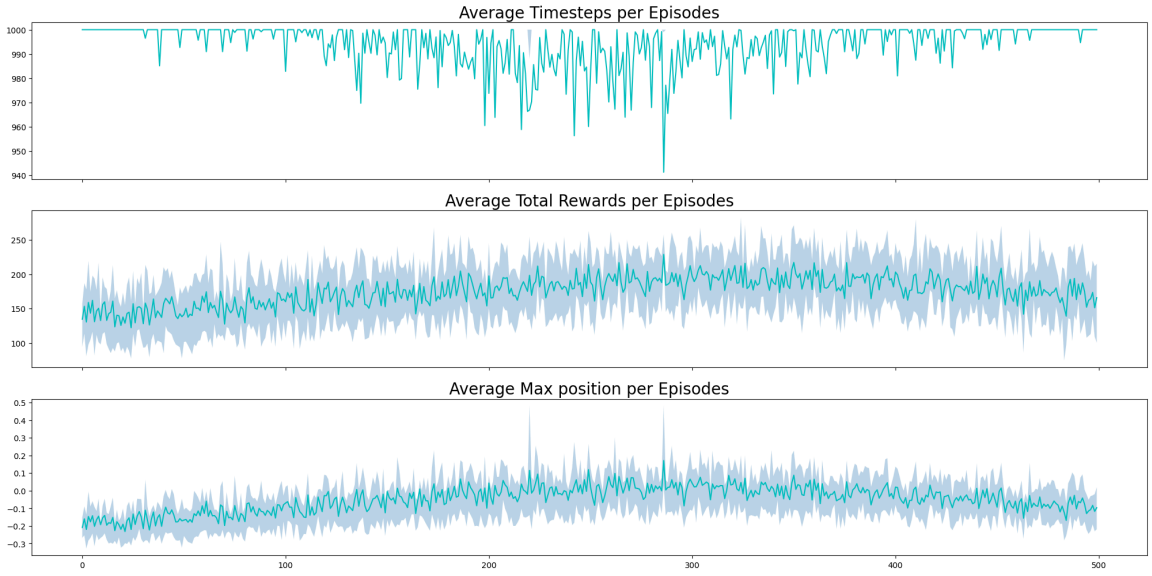


3.2. Actor Critic

Decaying-T

The following displays the time steps, total rewards, and max position per episode, averaged over 50 trials with $\alpha = 1e-4$, $\beta = 1e-4$, $\gamma = 0.5$, $T = 0.99$, PRF=1, NRF=0.1, SF=10, Decay Factor=0.99. As running 1000 episodes will take significantly longer, and the pattern emerges from early stages, we truncated the episodes to 500.

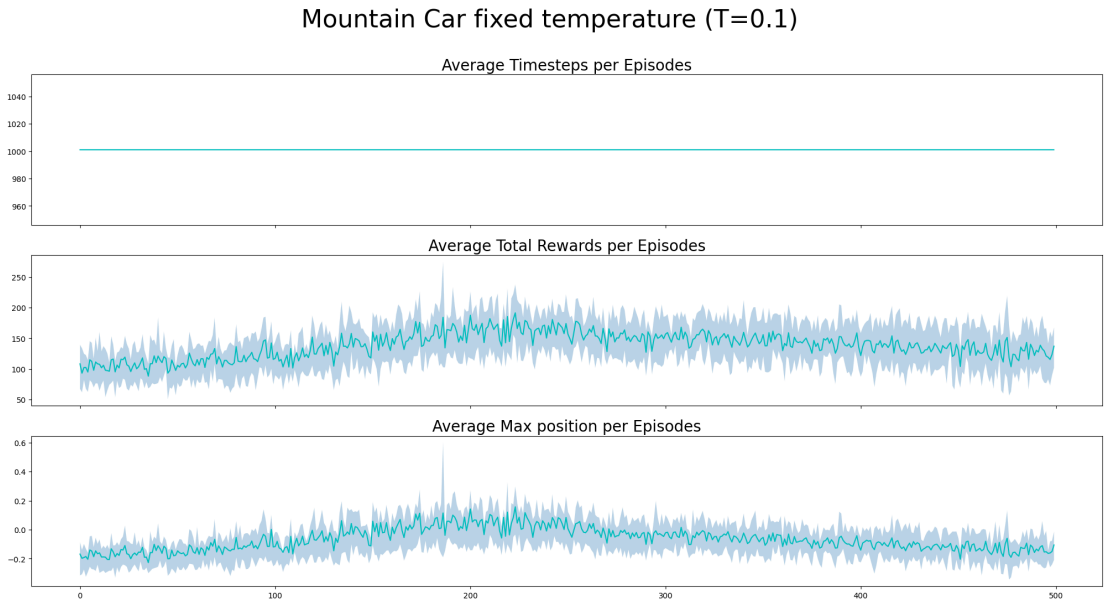
- The performance increases steadily over the episodes and starts to drop after 350 episodes
- The agent is able to consistently reach the top after 100 episodes



Fixed-T

The following displays the time steps, total rewards, and max position per episode, averaged over 50 trials with $\alpha = 1e-4$, $\beta = 1e-4$, $\gamma = 0.5$, $T = 0.1$, PRF=1, NRF=0.1, SF=5.

- As we can clearly observe, the agent is only making very mild progress and it fails to reach the top for all episodes.
- However, the agent still learned to move to the right as much as possible as the average max position of the agent is always greater than its origin (-0.4).



Comparing Fixed-T and Decaying-T

We achieve better results with decaying T compared with fixed T for both REINFROCE and Actor Critic. With deceaying T, the agent qcuikly learns to move towards the top of the hills, as measured by the max position, and starts to consistently succeed after around 75 episodes. With fixed T, the agent don't seems to learn much througout the episodes.

This is reasonable if we consider the essence of the problem. No matter how or where we start the episode, there is always a formulaic method to reach the top, that is to build up a consistent chain of actions (consistently accelerate to left, for example) to allow the agent to swing left & right, gain velocity, and eventually escape the valley . Therefore its better to apply the prior knowledge that the agent learned by acting greedily rather than acting randomly.

Moreover, If we choose to act more randomly (fixed T), we have a higher probability of breaking the chain of actions that we're building up (e.g accelerating to left when we almost making to the top), and thus, further preventing the agent to succeed within limiting time steps.

Comparing REINFORCE and Actor Critic

With fixed T, both algorithm failed to reach the top. However, its clear that Actor Critic learns better since it has a slow growing curve for both total reward and max position while REINFORCE's curve don't have any apparent pattern. Taking a closer look at the max position, we notice that Actor Critic's agent learns to move further away from the bottom of the valley than REINFORCE, showing the benefits of having a critic.

With decaying T, REINFORCE learns faster with agent strating to reach the top after 60 episodes than Actor Critic where the agent reaches the top after 100 episodes. In the early stages, the performance of REINFROCE increases intensely while the performance of Actor Critic increase more mildly. Nonetheless, Actor Critic's agent manages to persistently succeed at reaching the top longer than REINFORCE.

REINFORCE seems to be sensitive to episode number, achieving good performance only within a small range of episodes, while Actor Critic can achieve good performance for a longer time. Both REINFROCE and Actor critic suffer from a drop in performance.

Cite:
-Formula to stabalize discounted reward in REINFORCE taken from here.
-Wight initialization for policy net in REINFORCE taken from here.