RL Lab 4

Harsh Raj (180010017)
Akhilesh Bharadwaj (180010009)

Note: Code for all questions (including simulations etc) is attached in the submitted zip. Artifacts including the images generated are also attached in the zip. Please make sure to install dependencies mentioned in the requirements.txt before running any submitted code.

```
# Install dependencies
pip3 install -r requirements.txt

# run codes corresponding to each question
python3 runner.py
```

All the charts can be accessed online on this wandb repo.

Algorithms:

1. Monte Carlo:

Monte Carlo Control is an on-policy algorithm. It uses average episodic return to calculate the sample error to update the Q value. As it has to wait for the trajectory to finish, in order to be able to calculate the episodic return Gt, it is also classified as an offline algorithm.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

For the given three environments, we use the following hyper parameters for this algorithm:

- eps: 0.8 (To make it epsilon soft)
- decay_factor: 0.995 (after each episode, eps is multiplied by the decay_factor)

Since MonteCarlo uses episodic returns Gt to update its Q Values, the updates are less noisy. The demerit of this is that it makes the algorithm offline, ie. agent has to wait till the end of the trajectory in order to make any updates.

2. Q Learning:

Q Learning is an off-policy algorithm as it tries to approximate the optimal policy Q* while using the samples generated from current policy. The online version of Q Learning generally uses one step lambda return (TD-0) to calculate the sample error to update the Q value. Since the updates of Q value can be performed at each timestep, it is also classified as an online algorithm.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

For the given three environments, we use the following hyper parameters for this algorithm:

- eps: 0.8 (To make it epsilon soft)
- decay_factor: 0.995 (after each episode, eps is multiplied by the decay_factor)

Since Q Learning algorithm uses sample return at each timestep to update the Q Values, it is online, ie. agent does not have to wait till the end of the trajectory to update the Q Values.

3. SARSA:

SARSA is an on-policy algorithm. The online version of SARSA generally uses one step lambda return (TD-0) to calculate the sample error to update the Q value. Since the updates of Q value can be performed at each timestep, it is also classified as an online algorithm.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

For the given three environments, we use the following hyper parameters for this algorithm:

- eps: 0.8 (To make it epsilon soft)
- decay_factor: 0.995 (after each episode, eps is multiplied by the decay_factor)

- lr: 0.4
- gamma: 0.9

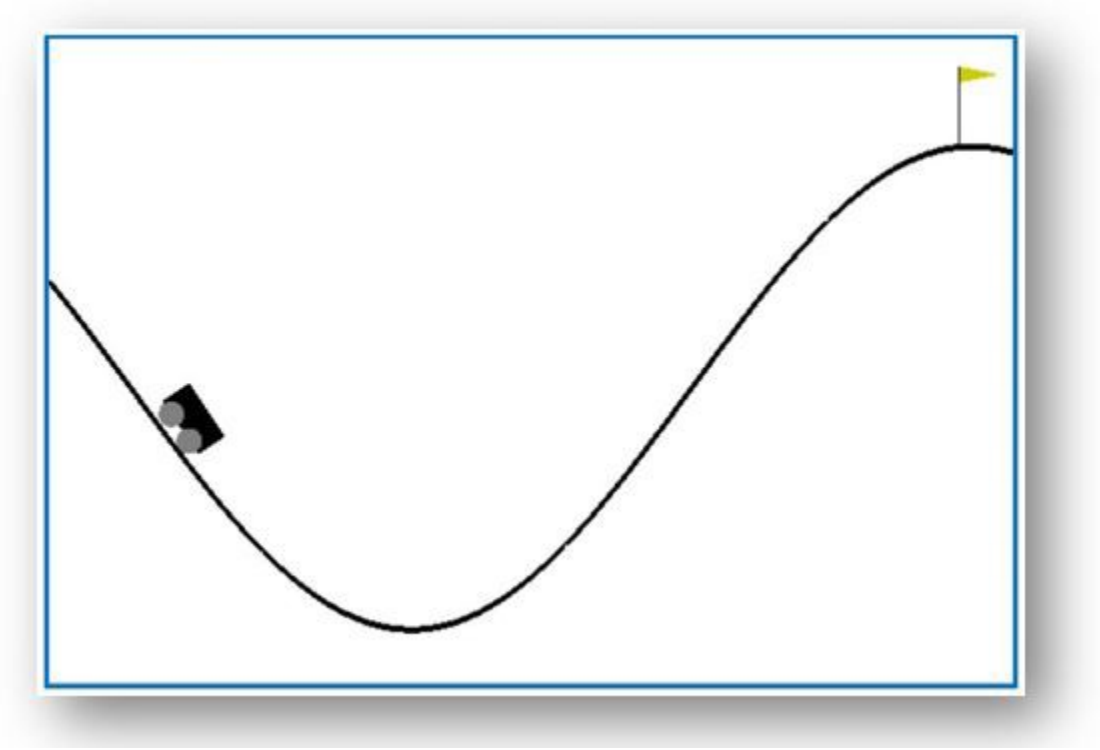
Environments:

1. MountainCar-v0

MountainCar-v0 is a standard gym environment having a discrete action space and a continuous observation space. The observation is given by the (position and velocity) tuple of the car at any instant. The agent can perform any of the following actions:

1. Apply thrust to the left
2. Apply thrust to the right
3. Do Nothing

The goal is to get the car at the goal state as shown in the image. The agent receives a reward of -1 at each instant, and a positive reward once it reaches the goal state. If the agent does not reach the desired state within 200 timesteps, the game is considered over.



2. Taxi-v3

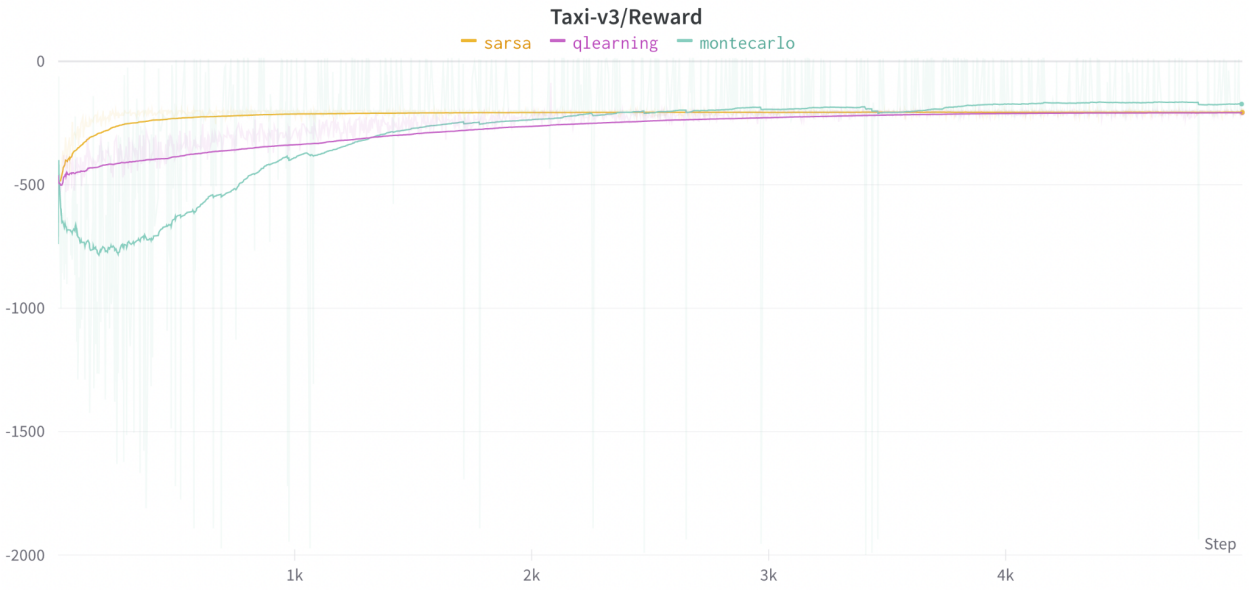
Taxi-v3 is another gym environment with discrete action and observation space. The goal of the agent is to pick the passenger from its source and drop it at the destination. The rewards associated with the environment are:
-1 per step unless other reward is triggered.
+20 delivering passenger.
-10 executing "pickup" and "drop-off" actions illegally.

3. SimpleLinear-v0

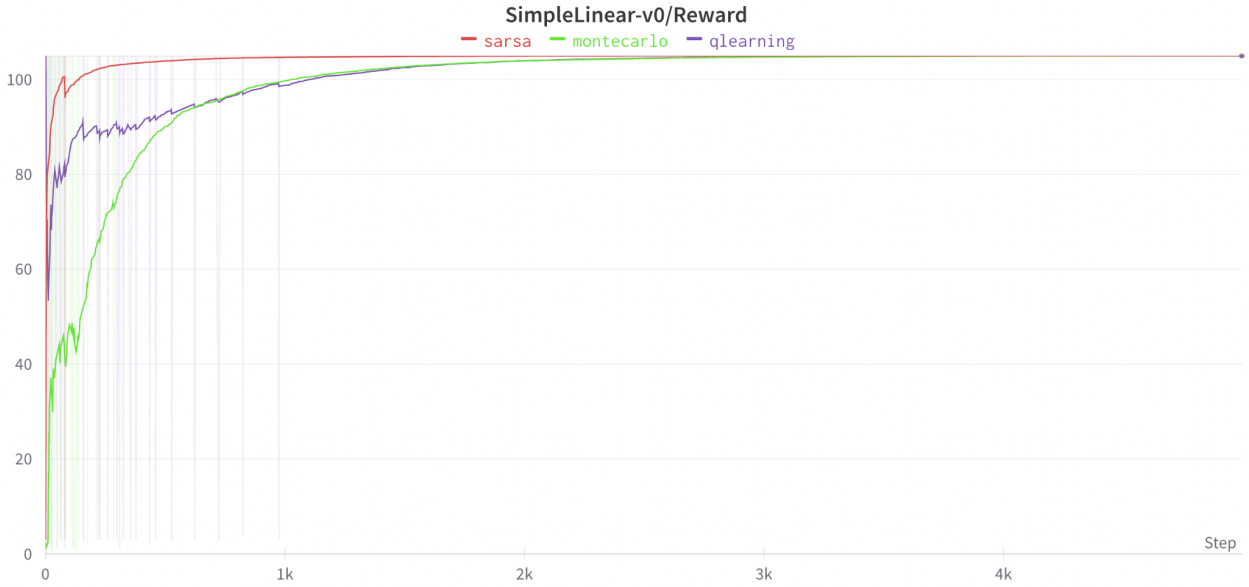
SimpleLinear-v0 is yet another simple environment having discrete state and action space. The agent starts at one of the states, and the goal is to reach the destination state. The game ends if the agent is not able to reach the destination within 10 timesteps. The reward distribution for the environment is:

- At each timestep, the agent receives a reward equal to the difference between the current state and the goal state.
- There is an additional reward for reaching the goal state.

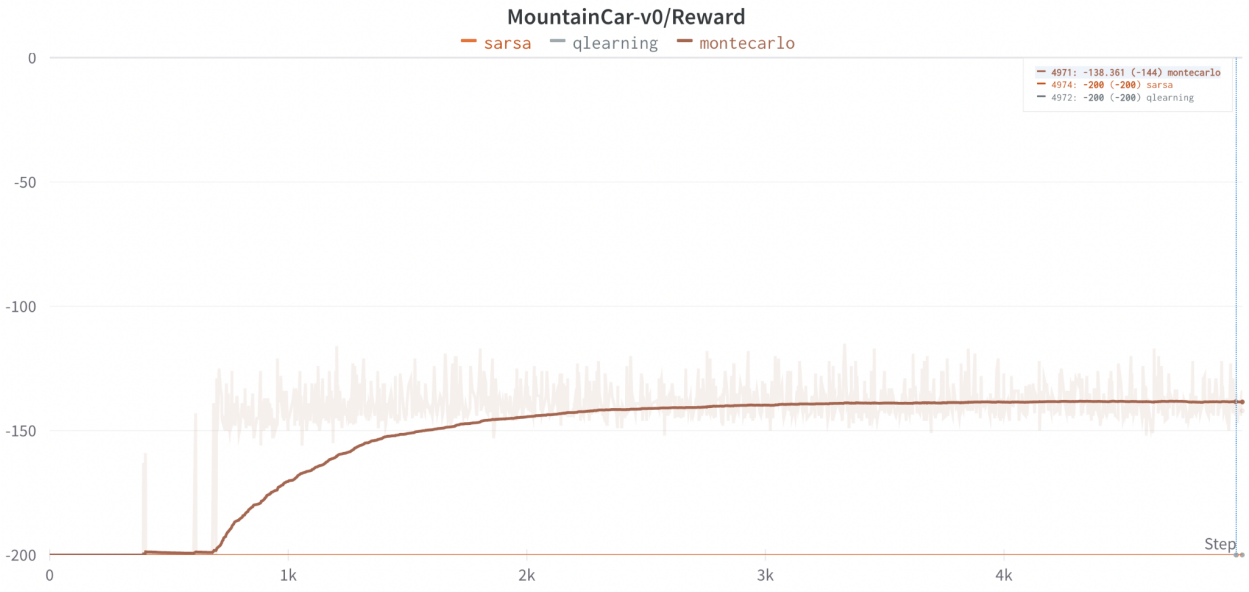
Observations and results:



For the task of Taxi-v3, we observe that all 3 algorithms converge to almost similar reward value after improving over the reward value it started with.



Similar trend is observed in the case of the SimpleLinear-v0 environment.



For the environment MountainCar-v0, the agents SARSA and Q Learning seem to be struggling. One of the reasons for it can be understood by the fact that the agents receive positive reward only after reaching the goal state. With significantly large state space, the probability of reaching the goal state randomly is pretty low.