Taxi Planning

Sidharth Agarwal 2019CS50661
Mohit Thakur 2019CS10373

November 2021

This exercise involves modeling a sequential decision-making problem. We consider the Taxi Domain problem that involves controlling a taxi agent that can pick up and drop passengers in a grid world. We will implement algorithms for arriving at a good policy offline using dynamic programming and later implement basic reinforcement learning methods for online learning.

Part A

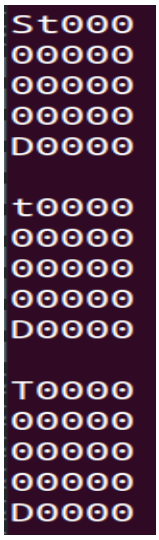
1.a

We define our model as follows

- **State space:**Our state space is of the form (locationOfTaxi, taxiHasPassenger, locationOfPassenger) Now this is because if the taxi doesn't have passenger it has to first reach it which is also based on location of passenger and if it has passenger then it has to only reach the destination.
- **Action Space:** The action space is (North, South, East, West, PickUp, PutDown)
- **Transition Model:** Each navigation action(North, South, East, West) succeeds with a probability of 0.85 and moves in a random other direction with probability 0.15. Whereas the pickup and putdown actions are deterministic. Now whenever the agent tries to move towards a wall it stays at the same location.
- **Reward Model:** A reward of (+20) is received when the taxi agent successfully delivers the passenger at the destination grid cell. Further, a reward of (-10) is received if the taxi attempts to Pickup or Putdown a passenger when the taxi and the passenger are not located in the same grid cell. Agent receives a reward of -1 for performing each other possible action.

1.b

We simulated the game on the terminal itself. We showed the progress of the game by showing 't' as the taxi without passenger, 'T' as the taxi with passenger, 'S' as the location of the passenger and 'D' as the destination of the passenger and the rest locations as '0'. Below we can see in first step the empty taxi goes to passenger and in second step it picks up the passenger.

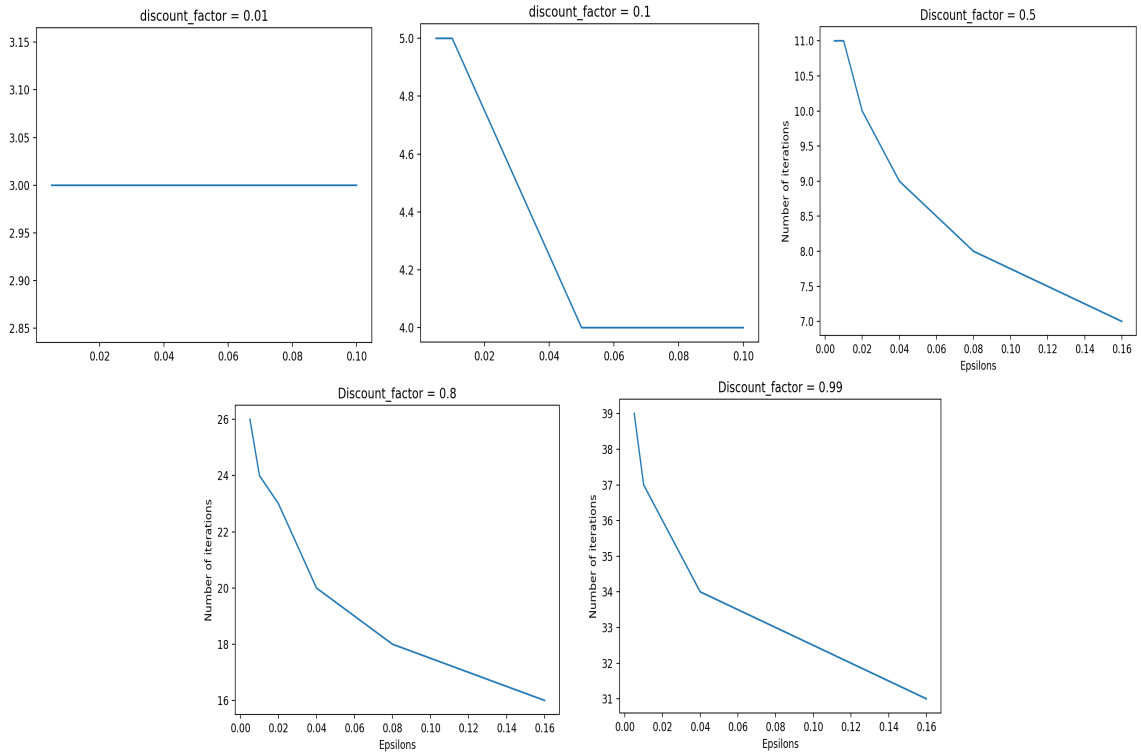


2.a

Here we can clearly notice that as we decrease the error allowed, i.e epsilon. The number of iterations increases. For discount factor as 0.9. This is because we can now terminate our algorithm even when their is a high error in utility from optimal utility.

Epsilon	Number of iterations
0.005	32
0.01	31
0.05	27
0.1	25

2.b



Here we can clearly see as the discount factor increases the sensitivity of number of episodes with epsilon increases. And the number of episodes for the same epsilon is also more for a higher value of discount factor. This is primarily due to more distribution of rewards across states which are far apart with higher discount factor. That is when we increase discount factor then the (+20) reward does not get discounted to much at far away places and hence it results in higher utility in each state so for the same epsilon it takes more iterations to get close to optimal higher utility values.

2.c

Now using Y (passenger initial location), G (passenger destination location) and R (taxi initial location). On the left we have first 20 state action pairs for discount factor to be 0.1 and in right we have it for discount factor to be 0.99.

```
0: (4,0)-->E
1: (4,1)-->W
2: (4,0)-->E
3: (4,0)-->E
4: (4,1)-->W
5: (4,0)-->E
6: (4,1)-->W
7: (4,0)-->E
8: (4,1)-->W
9: (4,1)-->W
10: (4,0)-->E
11: (4,1)-->W
12: (4,0)-->E
13: (4,1)-->W
14: (4,0)-->E
15: (4,1)-->W
16: (4,0)-->E
17: (4,1)-->W
18: (4,0)-->E
19: (4,0)-->E
20: (4,1)-->W
```

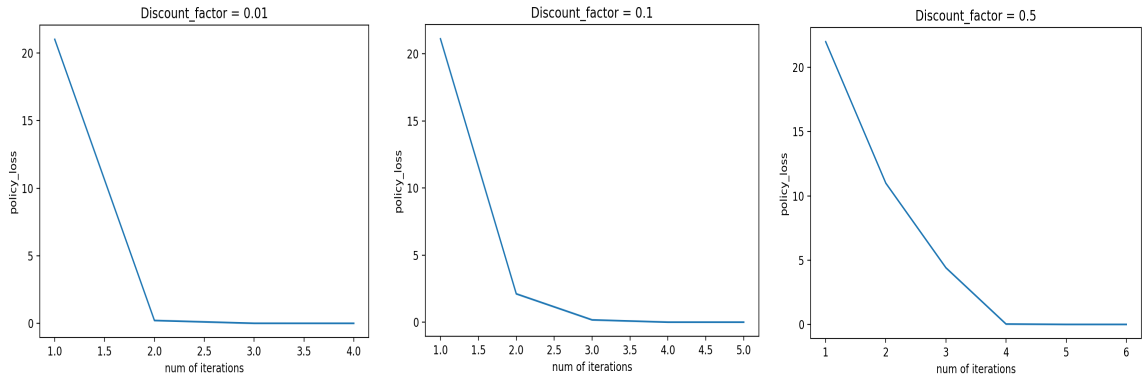
```
0: (4,0)-->S
1: (3,0)-->S
2: (2,0)-->S
3: (1,0)-->S
4: (0,0)-->PickUp
5: (0,0)-->N
6: (1,0)-->N
7: (2,0)-->E
8: (2,1)-->E
9: (2,2)-->N
10: (3,2)-->E
11: (4,2)-->E
12: (4,3)-->E
13: (4,4)-->PutDown
```

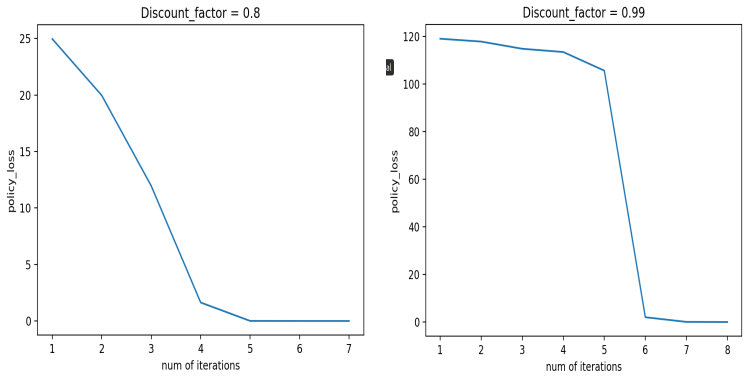
So it can be clearly observed that with very small discount reward it has not really learnt anything useful, this is due to the fact that (+20) reward is currently very far from the initial states so it doesn't matter much for the agent to get it since every action already takes -1 reward and even if the agent manages to reach (+20) reward it will receive a very very discounted version of that reward with low gamma. So since epsilon is not zero(or very very small) the small difference between utility of taking different actions is not large enough for the agent to learn any practically valid policy. Similar behaviour can be observed on different runs that for discount factor of 0.01, policy do not learn much about the path to follow.

3.a

Policy evaluation can be done using both iterative method and linear algebra method. Time complexity of linear algebra method is O(n^3) as it involves matrix inversion. While for the case of iterative method, time for one iteration in our implementation is O(n). In one iteration values can change in any manner and no such bound on steps required for eps error can be given. On contrast, linear algebra take fixed time for exact value. Method to be chosen can be determined by number of states, acceptable epsilon. Less number of states and very small epsilon will give us more preference toward linear algebra method. More acceptable epsilon, iterative method is preferred. Also we compared their times for this problem using discount factor to be 0.9 and epsilon to be 0.01. The iteration method took 1.274 seconds whereas the linear algebra method took 5.284 seconds.

3.b





Here we can see that the number of iterations taken increases with increasing discount factor and also the initial loss is higher in case of higher discount factor. As we have seen earlier, the optimal policy in case of small gamma is not very useful here for the problem statement because of single positive reward action and many negative reward actions. Hence for smaller gamma it doesn't really have to learn much and hence algorithm converges quickly.

Part B:

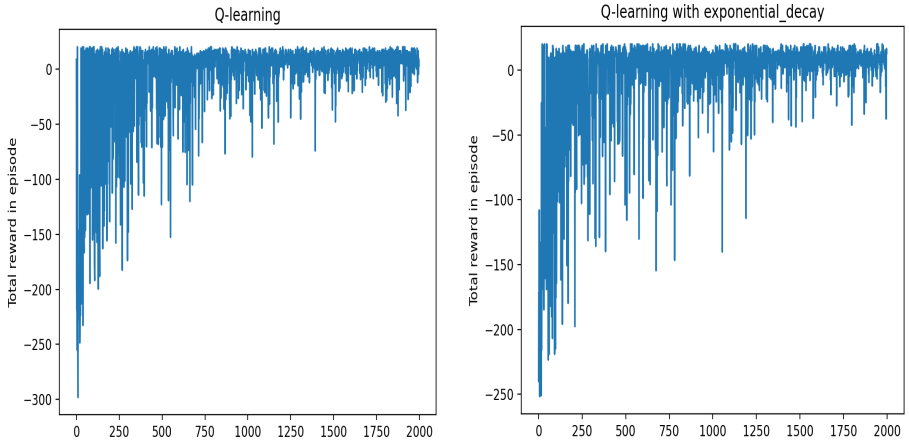
1)

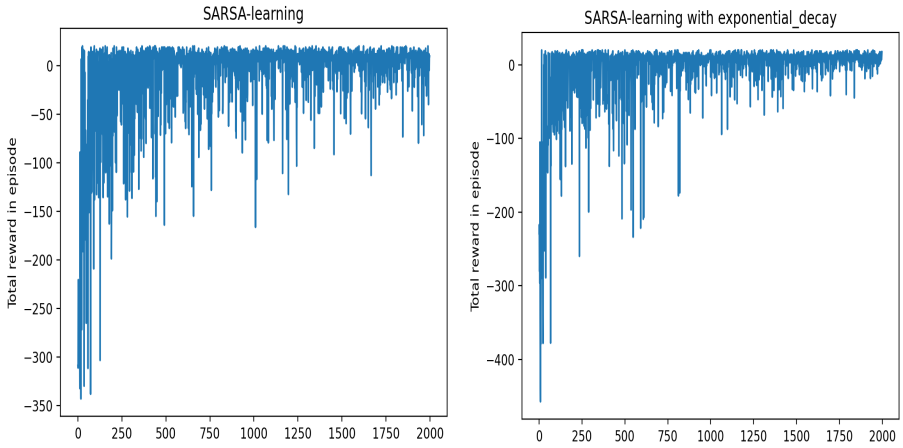
For exponential decay implementation we reduced exploration rate for each iteration and reset it for next episode. We used the function $epsilon_{new} = \frac{epslion_{initial}}{iteration_num}$ whenever their was an exponential decay.

2)

All the models were trained for 2000 episodes and up to 500 iterations. With a learning rate (alpha) as 0.25 and use a discount factor (gamma) of 0.99 and fixed exploration rate (epsilon) of 0.1. Now the average rewards that are being reported are averaged over 6000 runs

Algorithm	Average Discounted Reward
q learning	-9.36
q learning with exp decay	-15.34
sarsa learning	-18.35
sarsa learning with exp decay	-16.47





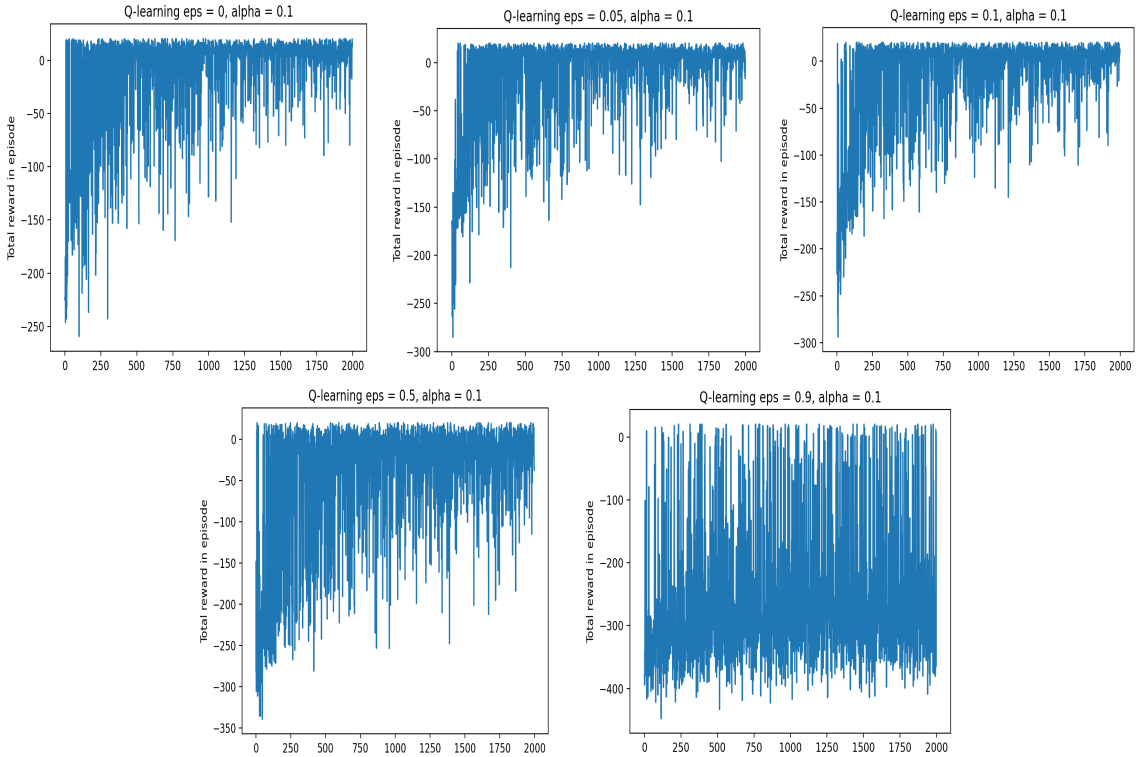
Although their was a huge variance observed in average rewards for each model but still in our case it was observed that simple q learning performs the best after 2000 episodes of training. Whereas if we observe the initial training episodes from 0-500 then it seems that sarsa learning with exponential decay seems to perform the best, because of its higher average rewards. So it seems sarsa learning performed better for the case of restricted training.

3)

Based on the highest average discounted reward it seems q-learning performed the best. On executing we can observe that the taxi in this scenario also acts almost optimally, with some deviations from the policy obtained by offline learning. As we know that online learners are robust to environment changes, the extent to which it learns makes it better choice than offline learner.

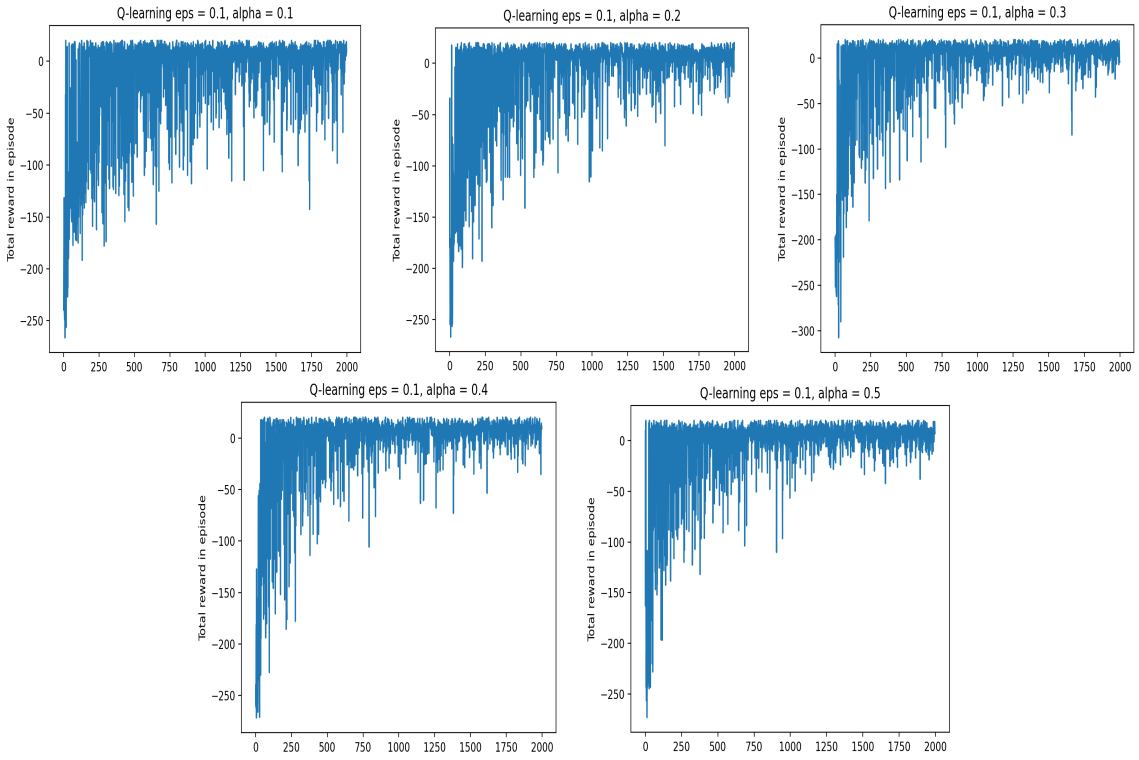
4)

Keeping the discount rate as 0.99 for all the experiments.
Now for learning rate set to be 0.1



Now we can clearly see that as we increase exploration factor, then it is taking much more episodes to increase the average reward per episode. And for eps = 0.9 it is not even learning anything, this is because most of the time it is acting randomly here and hence not learning anything.

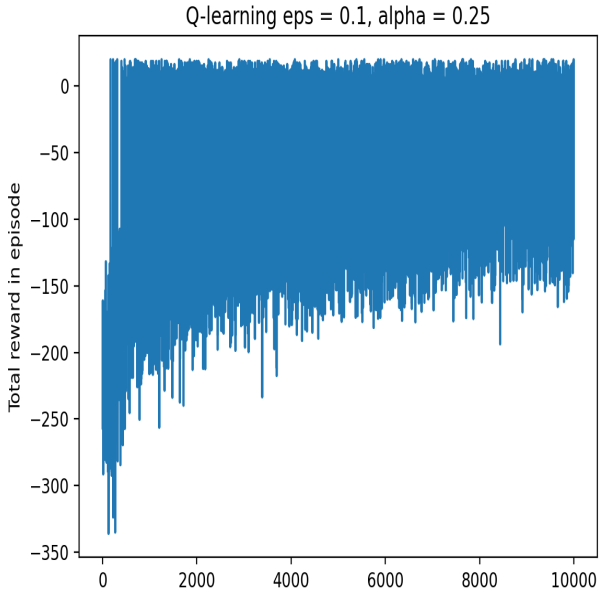
Now fixing the epsilon to be 0.1



Here we can see as we increase the learning rate the average discounted reward converge in shorter number of episodes. This is because if learning rate is very low then the model is not able to incorporate the new learned action much more and is still highly baised by previous actions which were randomly initialized at the first place.

5)

So for training the model we used 10000 episodes each with maximum iterations allowed to be 500. The learning rate used was 0.25, the epsilon value used was 0.1 and the discount factor used was 0.99.



The discounted reward averaged over 1000 episodes is -145.92.