COL333 A3

Planning in the Taxi Domain

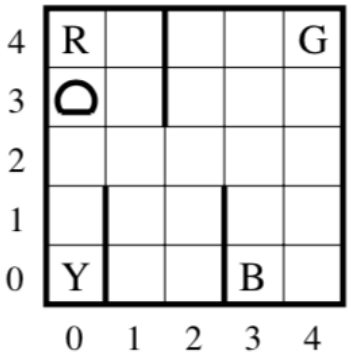
Aman Kumar - 2019CS10324
Gautambuddha N Meeshi – 2019CS10350

Code is written in the A3.py.

The code takes input for the question number. It also generate plots asked in sub-questions.

The code will first ask for inputs related to the part that has to be run and then it will ask for required data as input.

Part A: Computing Policies



1. Formulating the taxi domain as an MDP:

- A set (say S) of states: A state (say s) can be represented by the only three variables; location of the taxi, location of passenger and whether the passenger is picked up or not. We need this for checking a valid putdown action.
  Location of the destination are fixed for a particular instance, so we do not need to add this “constant” in our state.
  State s = (location of taxi, wheather passenger is pickedup or not, location of passenger)
  Eg: s = ((1, 2), True, (0, 4))
- A set (say A) of actions contains:

    - Four navigation actions that move the taxi one grid cell to the North, South, East or the West directions.
    - A Pickup action, where the taxi attempts to pick up the passenger.
    - A Putdown action, where the taxi drops off the passenger.
- The transition model:
    - Each navigation action succeeds with a probability of 0.85 and moves in a random other direction with probability 0.15. Movements that attempt to cross the boundary of the grid or a wall results in no change in the state.
      Let T is the transition function and T(s, a, s') represents the probability that action a from state s leads to state s' and passenger is already picked up.
      Example:

      $$T(\{2, 1, ..\}, N, \{2, 2, ..\}) = 0.85$$

      Here it represents the transition from state {2, 1, ..} ((2, 1) is the coordinates x and y of the location of the taxi) to state {2, 2, ..} by moving in North(N) direction. Since this transition is valid (not a movement to cross the boundary of the grid or a wall) and North to {2, 1} is {2, 2}, it will succeed with a probability of 0.85. The taxi will move in a random other direction with the probability of 0.15. Since the other direction will be random and there are 3 other directions:

      $$T(\{2, 1, ..\}, N, \{1, 1, ..\}) = 0.05$$
      $$T(\{2, 1, ..\}, N, \{2, 0, ..\}) = 0.05$$
      $$T(\{2, 1, ..\}, N, \{3, 1, ..\}) = 0 \text{ (Wall)}$$

      In the transition T({2, 1, ..}, N, {3, 1, ..}) the texi will not be able to reach the state {3, 1} because of the wall and it will remain in the same state i.e. T({2, 1, ..}, N, {2, 1, ..}) = 0.05.

    - Picking up the passenger and Dropping is also in transition function.
      T(s, P, s').
- Reward model:
  The taxi agent receives a reward of (-1) for performing each action. 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.

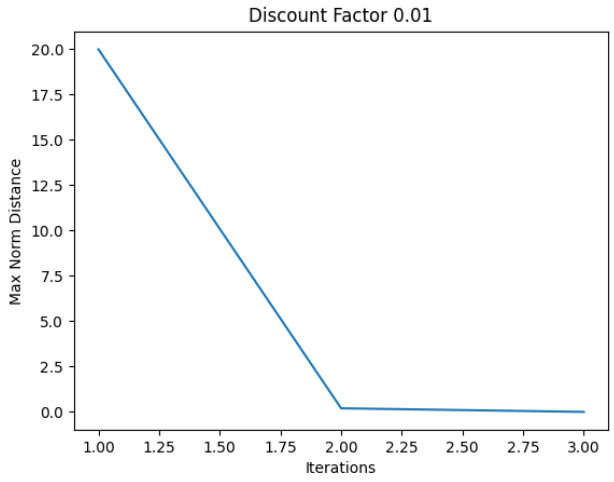
  For part A1b, we have implemented a simulator which take starting depot for the passenger, a different destination depot, and a starting location for the taxi in the grid as input and simulate all the steps taken

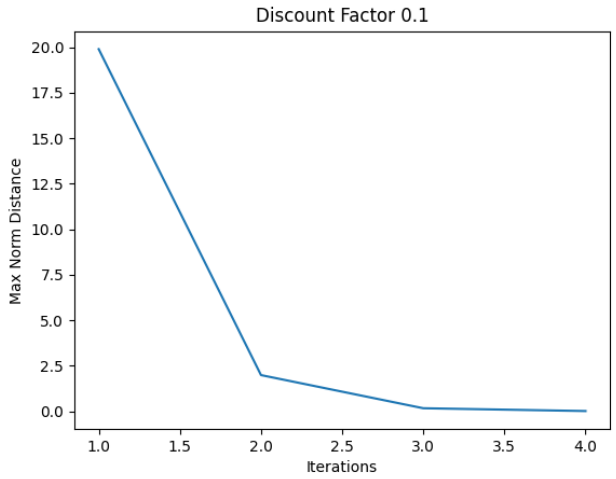
by AI. We implemented a function to calculate the next state given the current and action taken.

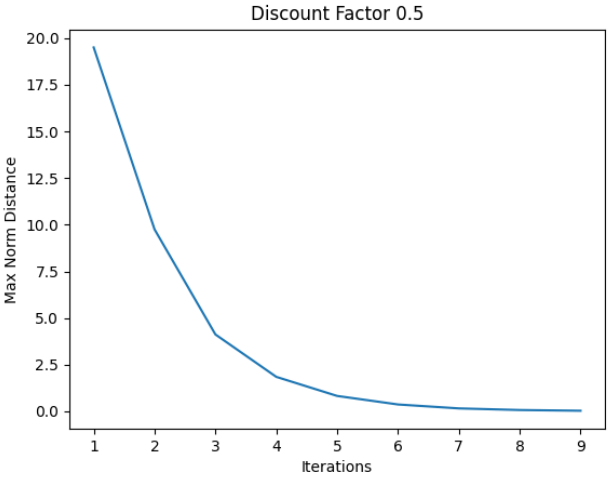
2. Implementing Value Iteration for the taxi domain.

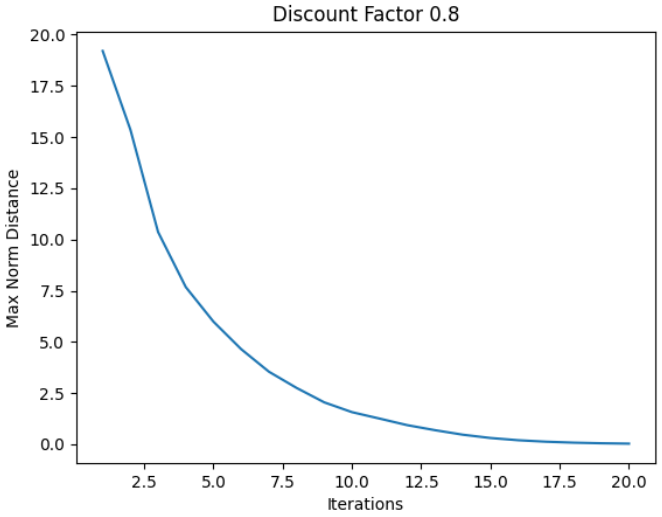
a. We have implemented the value iteration function and on choosing a discount factor of 0.9 and epsilon = 0.05, the code is taking a total of 34 iterations.

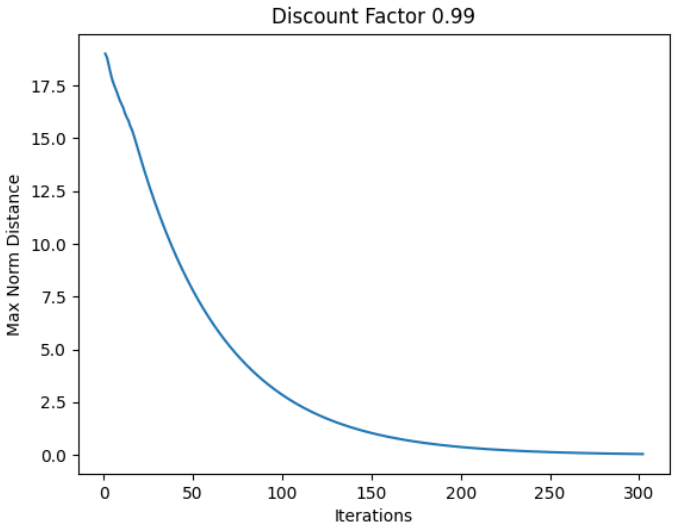
b. Plots:






Discount Factor 0.5
Max Norm Distance
Iterations
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1
2
3
4
5
6
7
8
9


Discount Factor 0.8
Max Norm Distance
Iterations
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Discount Factor 0.99
Max Norm Distance
Iterations
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
50
100
150
200
250
300

What we observed from these plots is that for a large discount factor, the value iteration algorithm is taking a lot of iterations. Larger the discount factor is, the larger the total number of iterations.
The reason for that is that the discount factor controls the importance of future rewards. For small discount factors, the dependence of the utility values on the future reward will be very less. For every next future reward, the discount factor keeps multiplying, and eventually the future reward stop affecting the utility values. Because of very low changes in each iteration, the algorithm terminates.

c. We first tried the example:
    Passenger initial location - (0, 0)
    Passenger destination location - (4, 4)
    Taxi initial location - (0, 3)

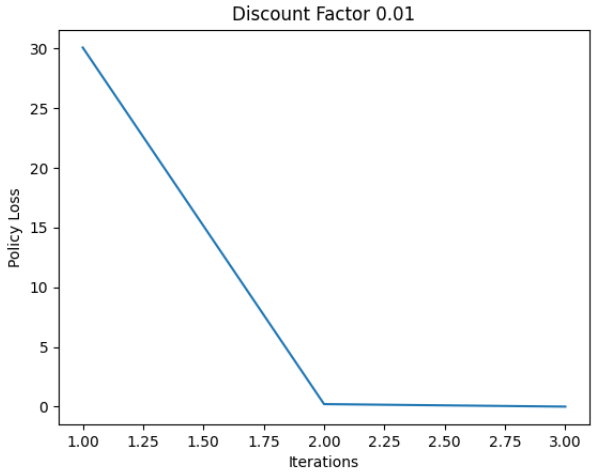
   For gamma = 0.1, It choose some good choices initially and picked up the passenger in 6 steps. After that, it got stuck there and it keep calling pickup action. Total reward = -20.
   For gamma = 0.99, it made good choices till the end. Picked up the passenger in step 5 and dropped it to the destination location in step 15. Total rewards = 6
   We tried varying these three locations a few times but every time for gamma = 0.1, it was unable to finish the instance. Usually, the taxi picks up the passenger and keep calling the pickup action or taxi go to the destination location and try to drop the passenger without even picking it up.

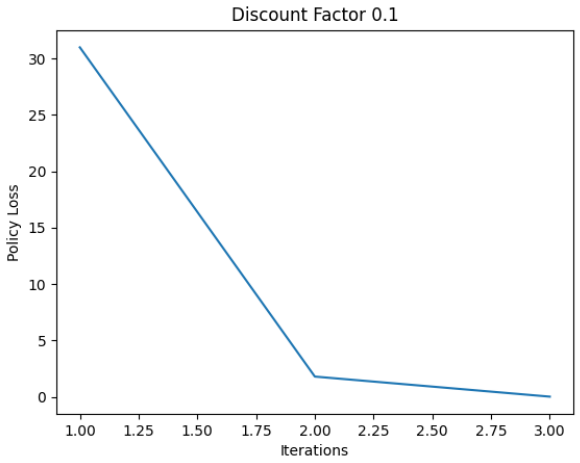
   This is happening because due to small gamma (0.1), the big reward (droping of a passenger at the correct location) ends up carrying less weight as gamma keeps multiplying.

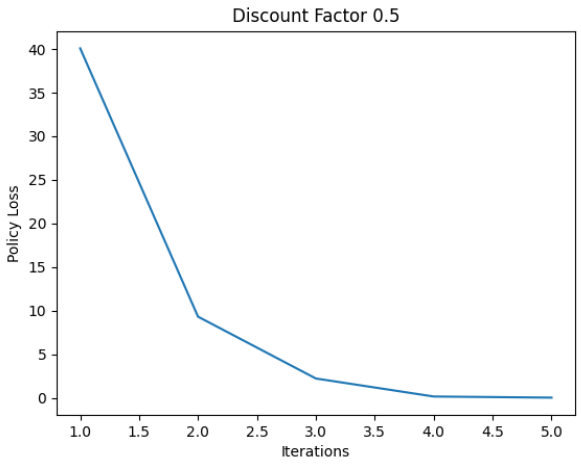
3. Implementing Policy Iteration for the problem

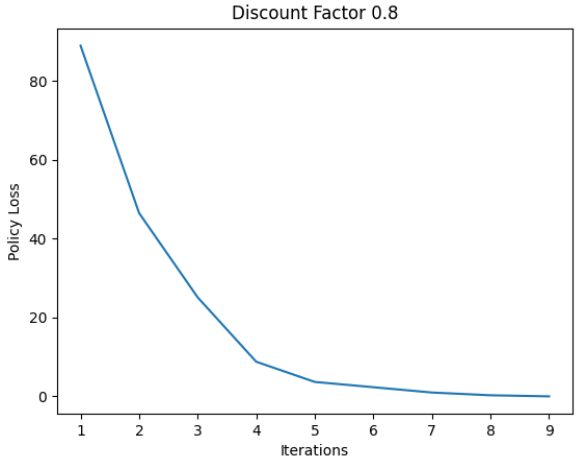
a. For a small number of states, it is better to use the linear algebra method as it gives the optimal solution. But in the case of a large number of states, the linear algebra method takes a lot of time and is hence not solvable by a normal computer. We tried using the linear algebra method in our policy iteration function but it was not able to run in time.

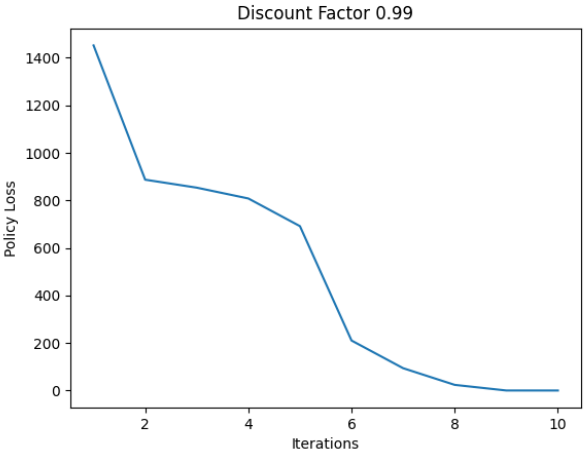
b. Plots:


Discount Factor 0.01
Policy Loss
Iterations
30
25
20
15
10
5
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Discount Factor 0.1
Policy Loss
Iterations
30
25
20
15
10
5
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Discount Factor 0.5
Policy Loss
Iterations
40
35
30
25
20
15
10
5
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0





What we observed from these plots is that for a large discount factor, the policy Iteration function is taking a lot of iterations. We can also see that for a large value of gamma, the policy loss is also large. This is because for large value of gamma, we give more weight to future rewards than of small value of gamma. Due to which the change in a value is higher in each iteration.

We can apply the same logic that we did for the value iteration problem.

Another thing we noticed is that for a small value of gamma it is taking a lot of time to converge to the optimal state. Here the values are converging faster than the policies.

Part B:

1. Note: this part code requires matplotlib

Code is written in the A3.py.

The code takes input for question number, sub question in part B and generates plot and terminal output after executing the particular part.

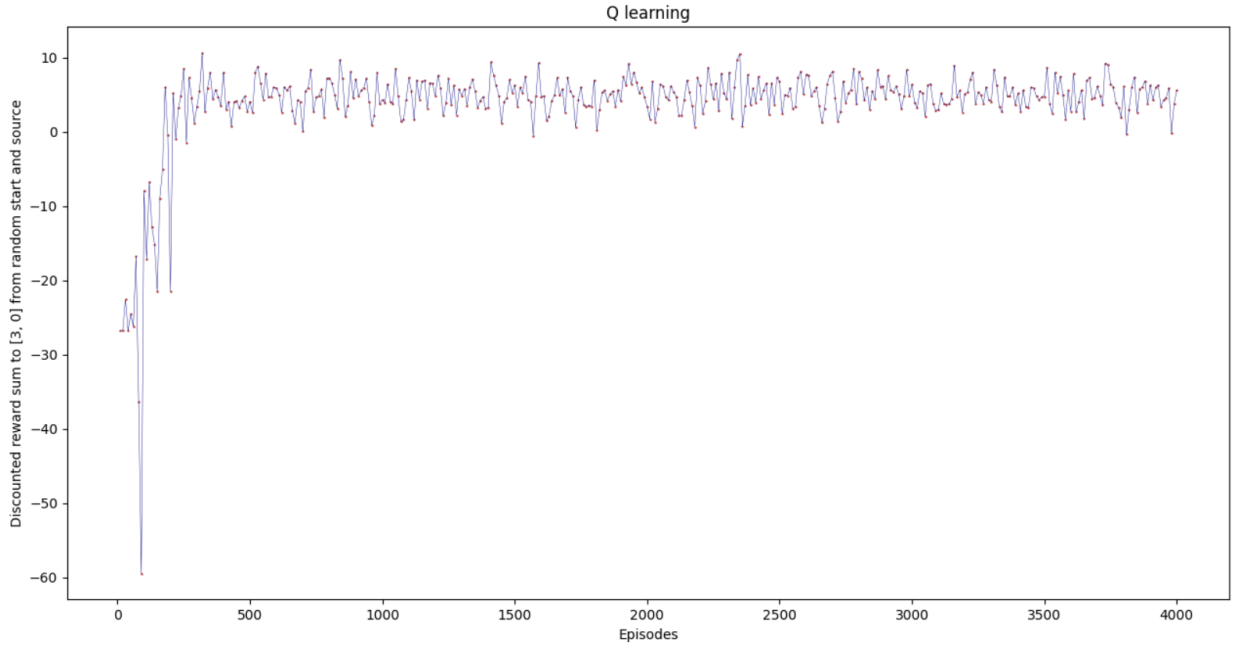
2.

Each algorithm runs for 4000 episodes, till 500 steps or a correct drop at destination. Destination is fixed to Blue (3,0). The taxi location and passenger depo are taken randomly in each episode. To make the comparison the random function is seeded with same seed for all the algorithms.

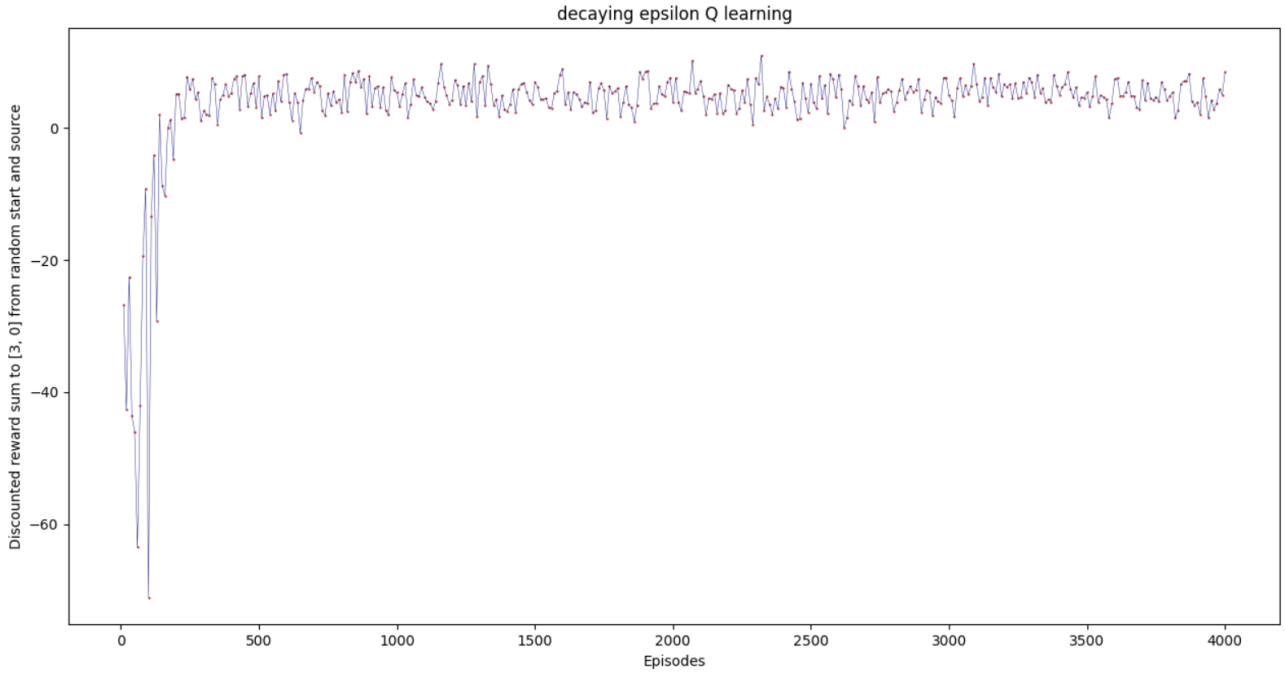
The epsilon is decayed exponentially:

$\epsilon = \epsilon_0.\exp(-0.00001*\text{iterations})$; iterations – is the no. of Q times update equation is executed.

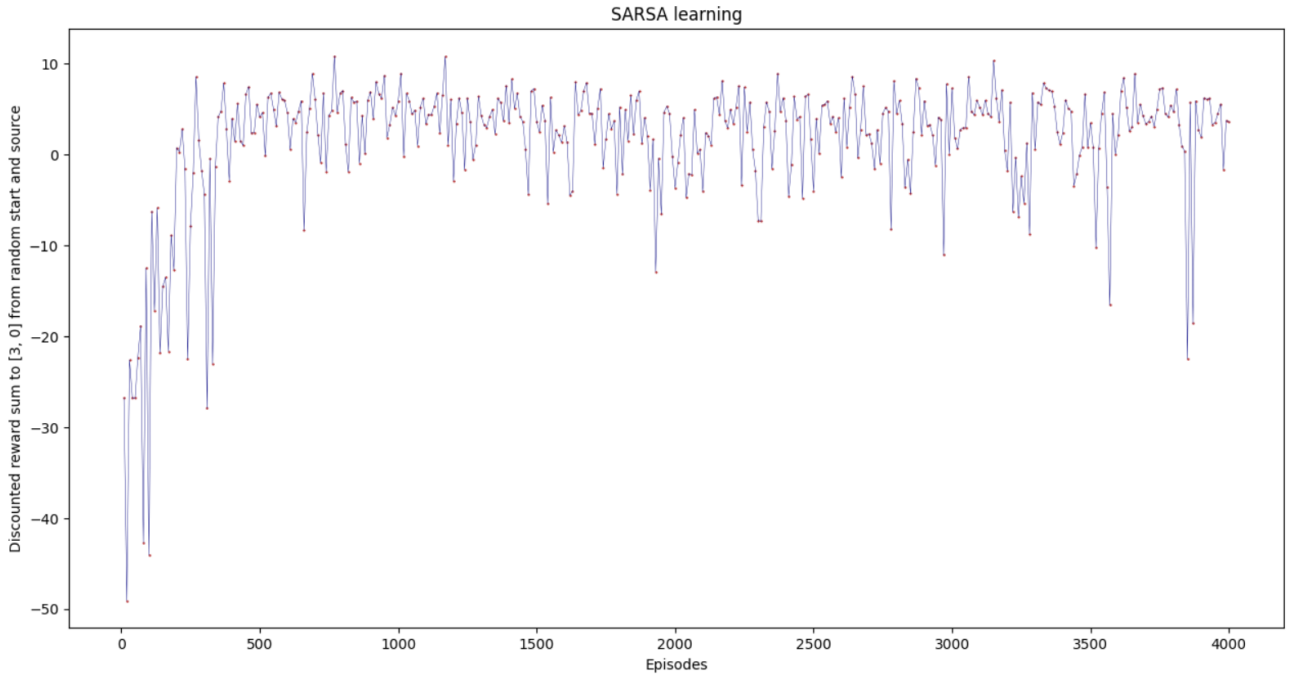
Q learning



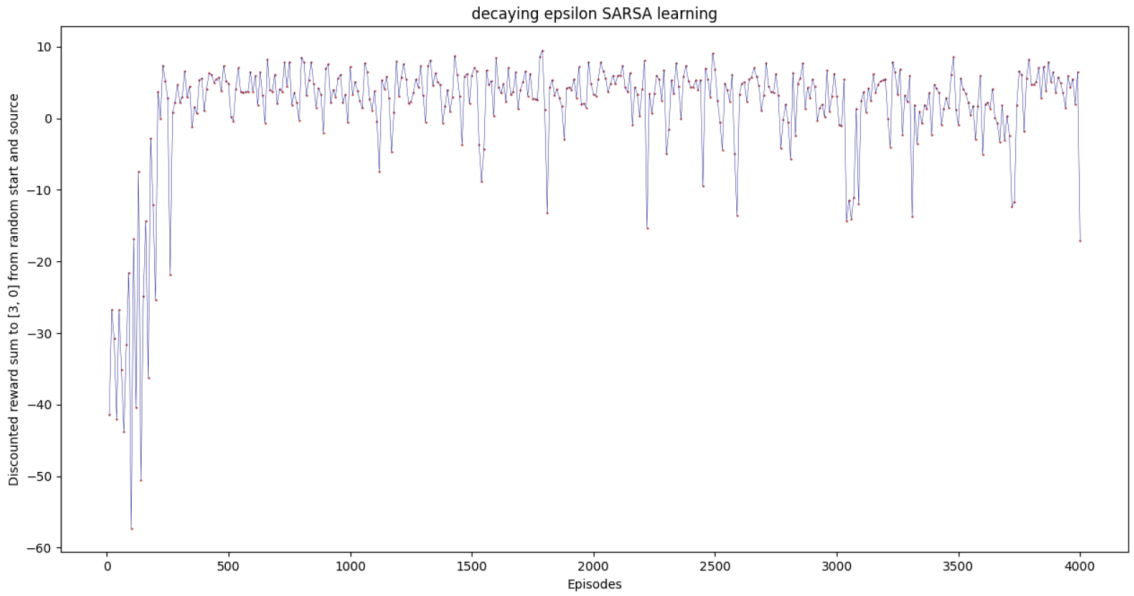
Decay Q learning



SARSA



Decaying epsilon SARSA



Observation –

Algorithm	Total pickups	Total drops	Correct drops	Max discounted reward sum
Q learning	4572	4571	3967	10.652
Decaying ϵ Q	4450	4447	3959	10.983
SARSA	4945	4943	3964	10.846
Decaying ϵ SARSA	4648	4644	3958	9.491

All the algorithms converge within 500 episodes.

The Q learning clearly shows faster convergence and lesser variation compared to SARSA. This is because the SARSA takes action in next state with ϵ randomness whereas the Q learning always takes the best possible next Q value. So in SARSA it may pick a next action having very low Q value during random sampling.

Highest discounted reward is obtained in Q learning with decaying epsilon. The ratio of correct drops to total drops is also maximum for it. This may indicate that random moves are important only in the initial phase of learning, once a path is found the randomness disturbs the optimal policy.

The optimum value of decay constant differs for Q learning and SARSA very much because the effect of randomness is higher in SARSA.

3) Checking the policy for 5 different pairs of taxi, passenger location.

Destination is fixed to Blue (3,0). Q learning with exponentially decaying epsilon is used.

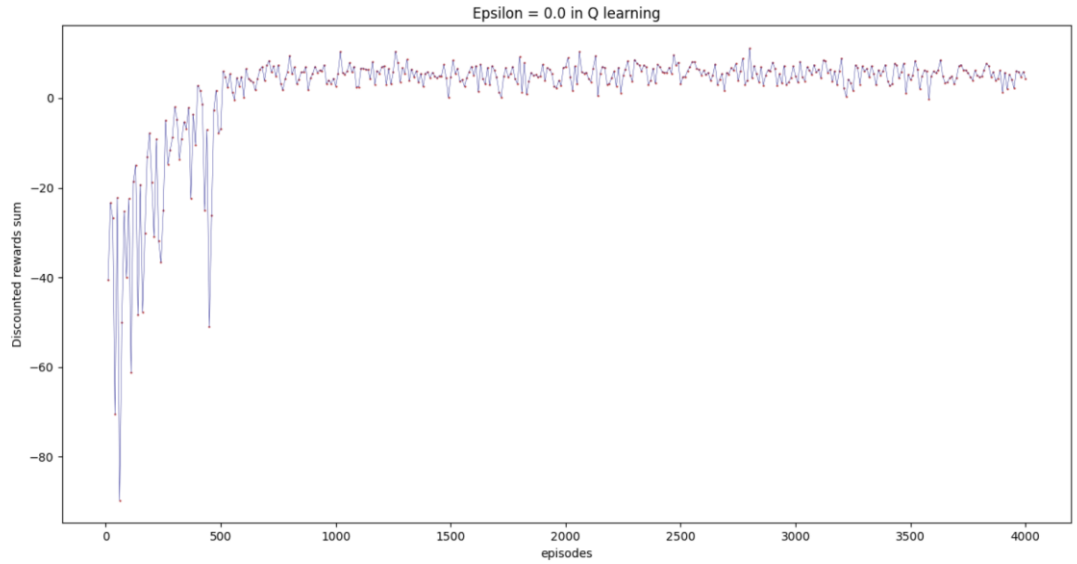
Observation –

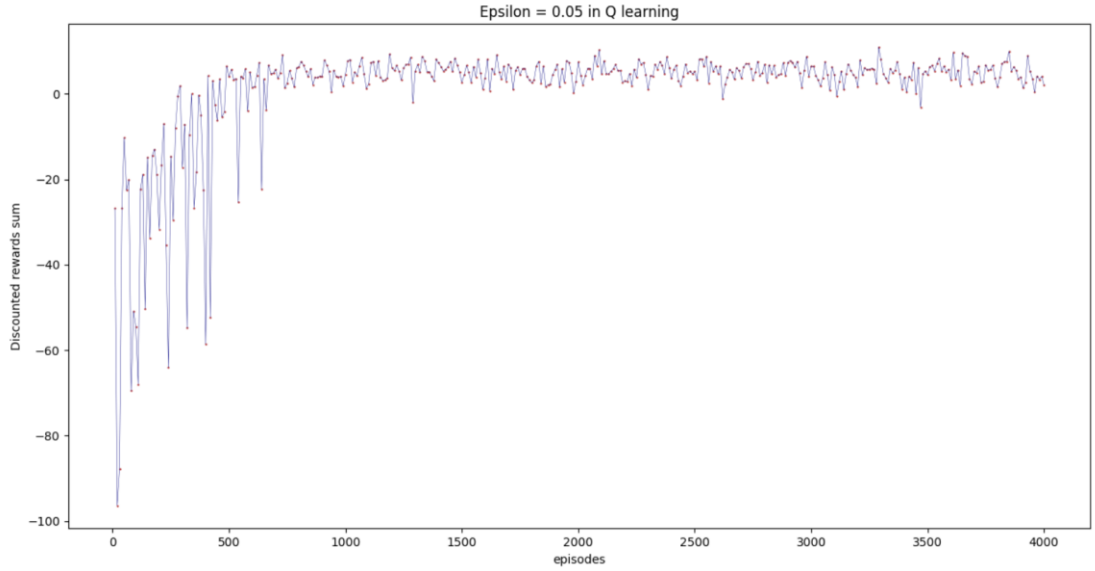
Each execution performs exactly 1 pickup and 1 drop thus policy is very optimal in identifying the location of passenger and destination.

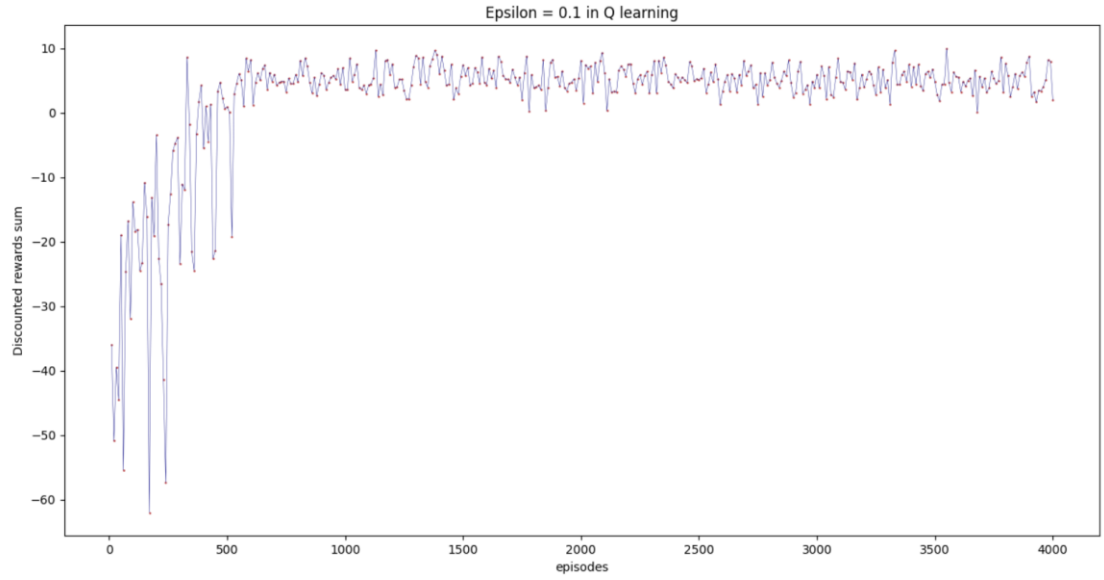
Taxi start	Passenger	Minimum steps required	No. of steps	Discounted reward sum
(4,4)	Y (0,0)	17	21	-1.85
(2,2)	R (0,4)	13	17	2.18
(0,0)	G (4,4)	15	15	4.25
(1,4)	G (4,4)	14	14	5.30
(3,1)	Y (0,4)	15	21	-1.85

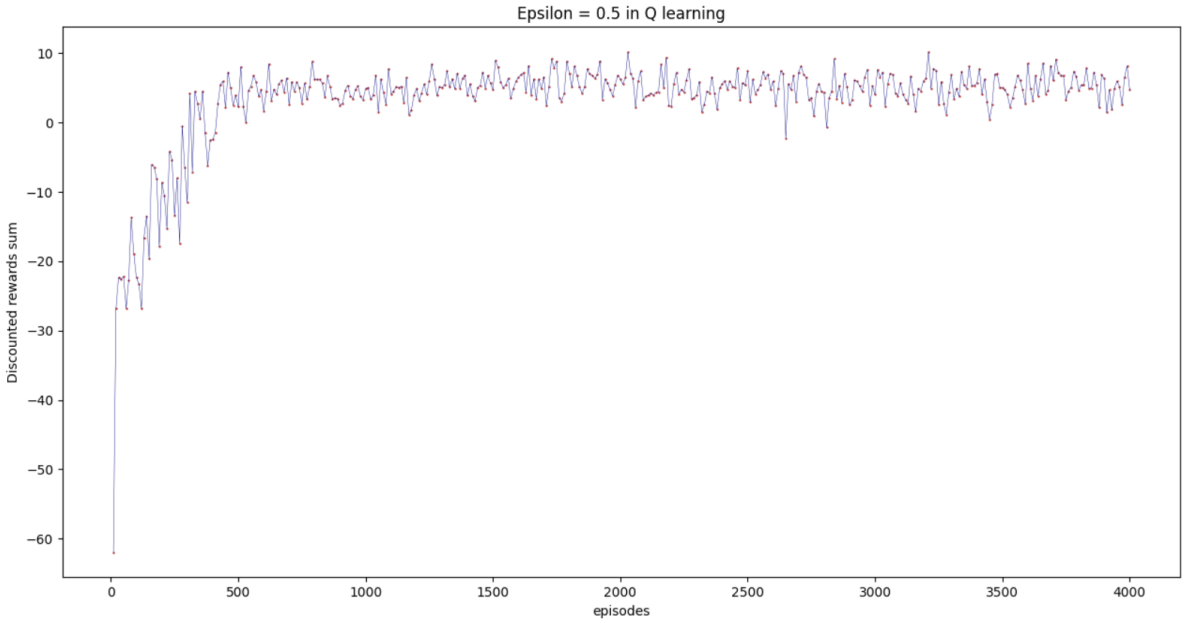
1. States requiring more steps have lesser discounted reward sum since each step contributes -1 except the correct drop (+20).
2. The no. of steps is minimum when the passenger and destination location are nearer because lesser random actions are taken. And the information is higher near the destination.

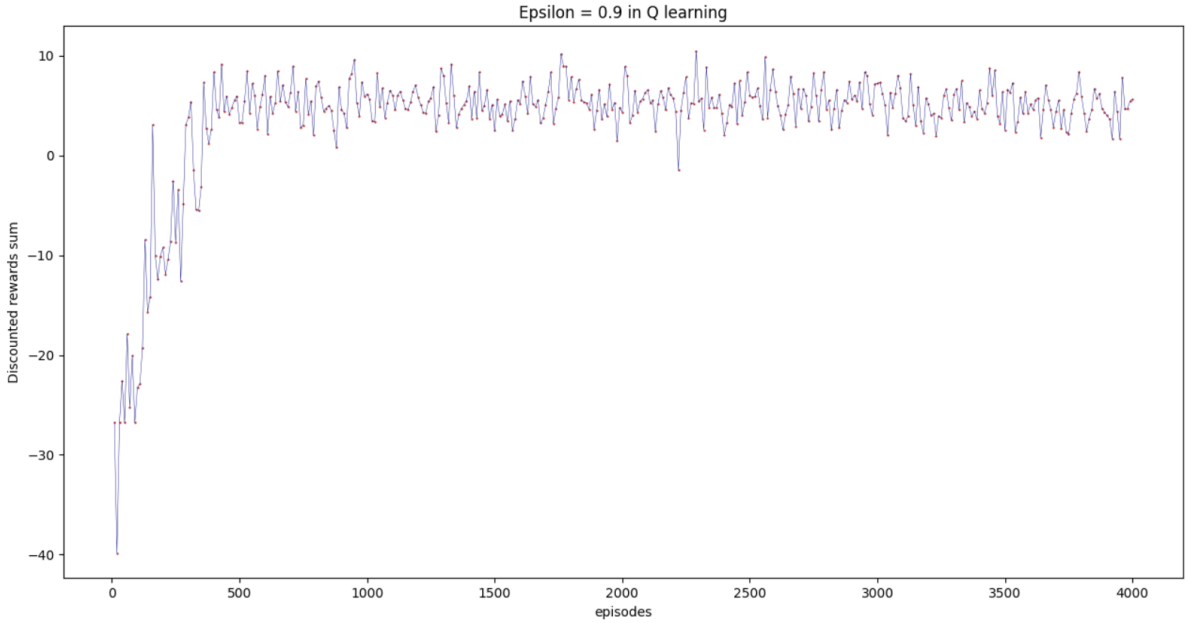
4) Varying the exploration rate, keeping learning rate = 0.1











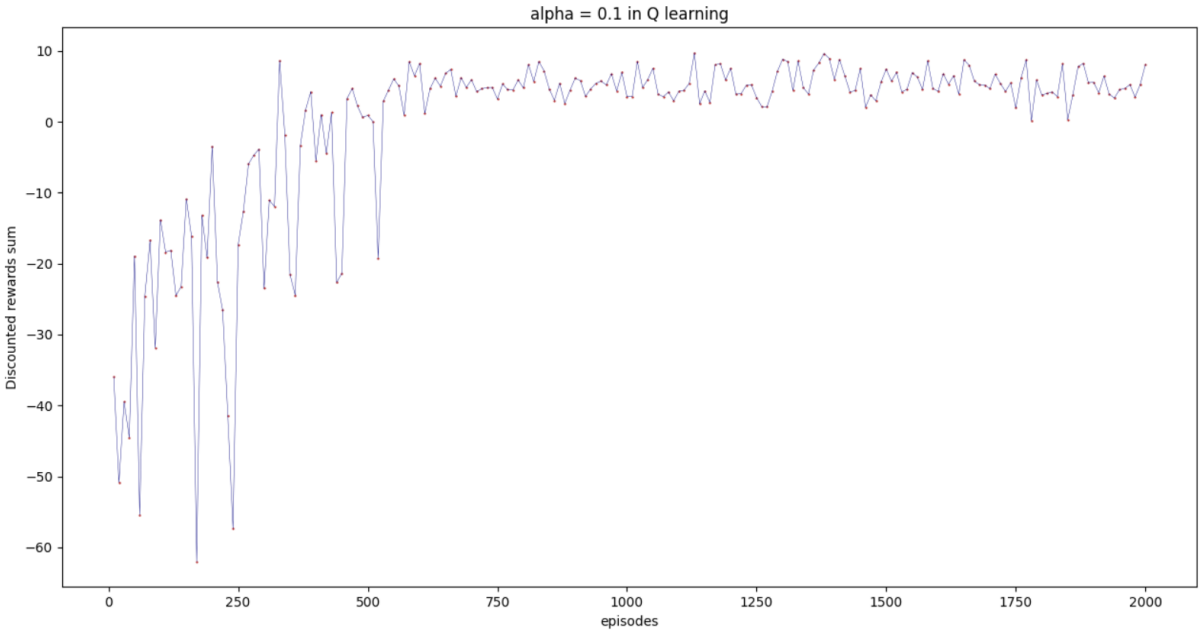
Epsilon	Pickups	drops	Correct drops	Max expected reward sum from start
0.0	4300	4298	3929	11.159
0.05	4486	4482	3938	10.941
0.1	4688	4686	3942	9.963
0.5	9097	9091	3908	10.191
0.9	29620	29345	2000	10.451

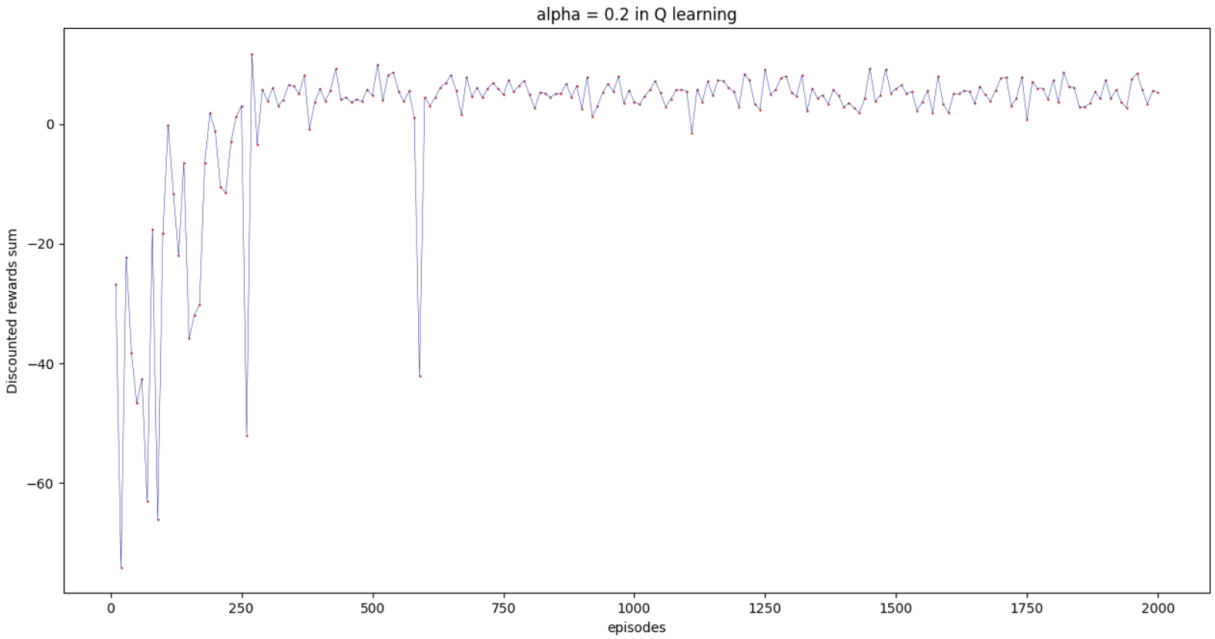
Observation –

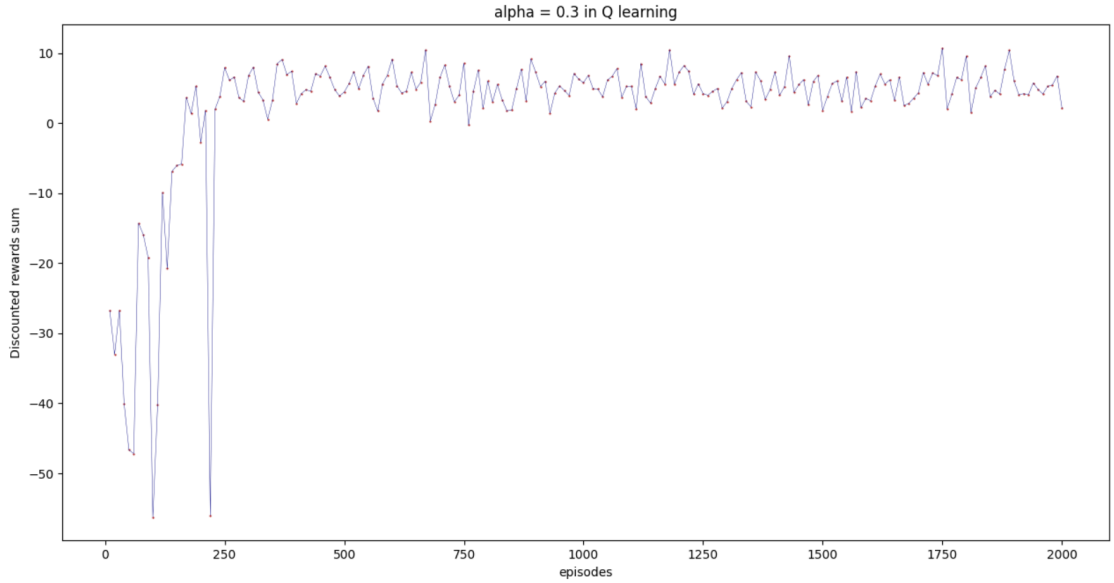
1. Increasing exploration decreases correct drops significantly. For epsilon = 0.9 the correct drop has reduced to 2000 (half of total episodes). This is because random actions are taken very often and the agent rarely acts according to the policy.
2. The learning is faster with increase in Epsilon (the graph converges with lesser episodes), meaning the exploration finds the path quicker when the destination location is not yet known.

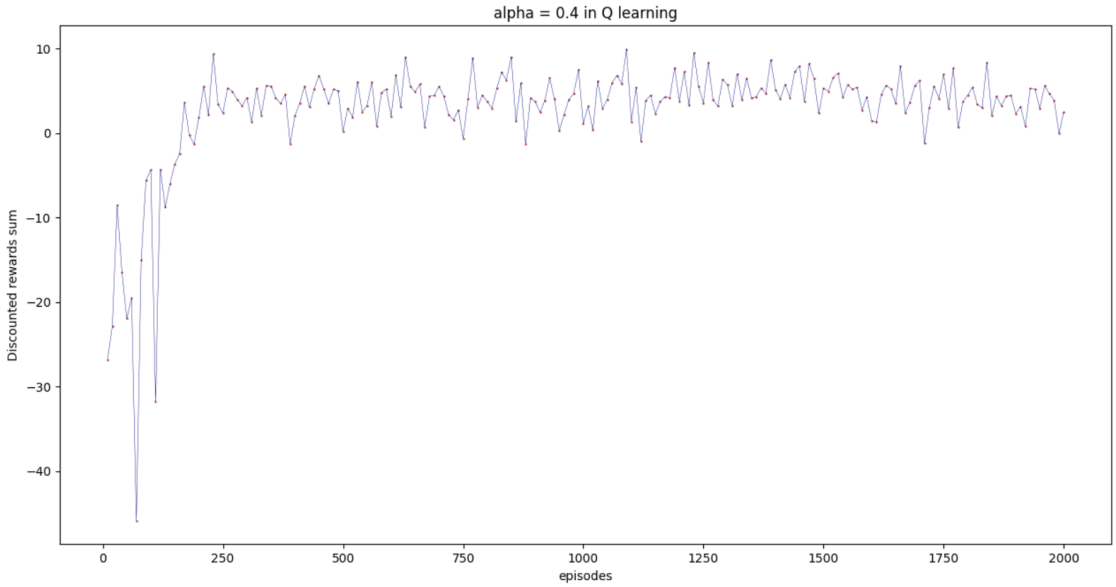
3. Increase in epsilon increases the variance of reward sum significantly. Because more random actions are taken leading to negative reward.
4. Time taken increases with increase in exploration rate, due to random movements the destination is reached late

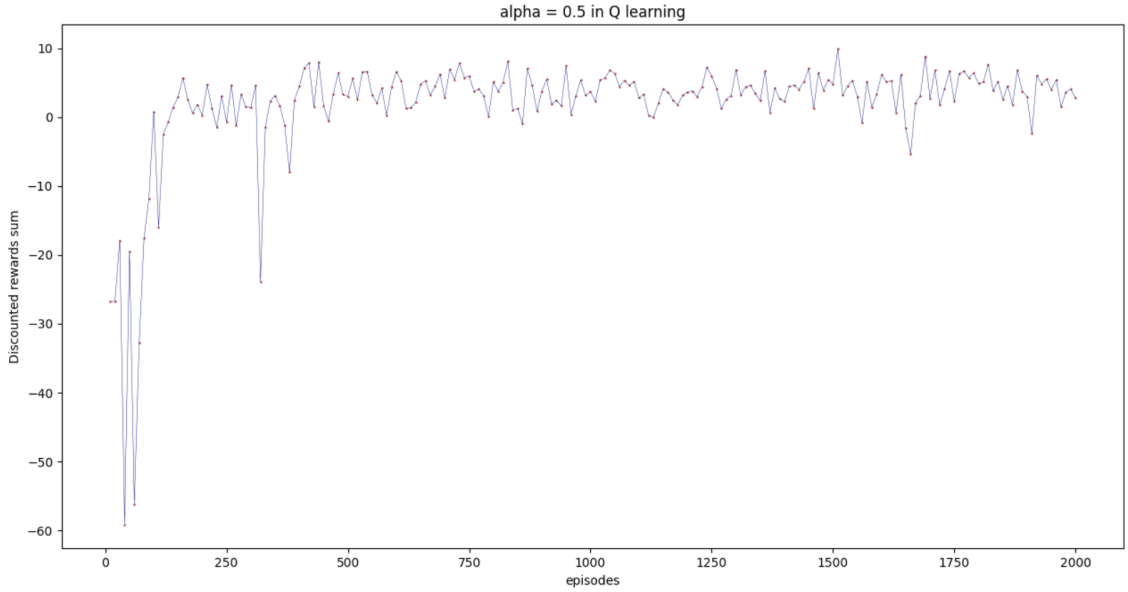
4) Varying learning rate keeping exploration rate = 0.1






alpha = 0.3 in Q learning
Discounted rewards sum
episodes


alpha = 0.4 in Q learning
Discounted rewards sum
episodes


alpha = 0.5 in Q learning
Discounted rewards sum
episodes

alpha	Pickups	drops	Correct drops	Max discounted reward sum from start
0.1	2439	2437	1942	9.727
0.2	2409	2409	1947	11.679
0.3	2387	2383	1960	10.732
0.4	2422	2422	1969	9.948
0.5	2373	2371	1970	9.949

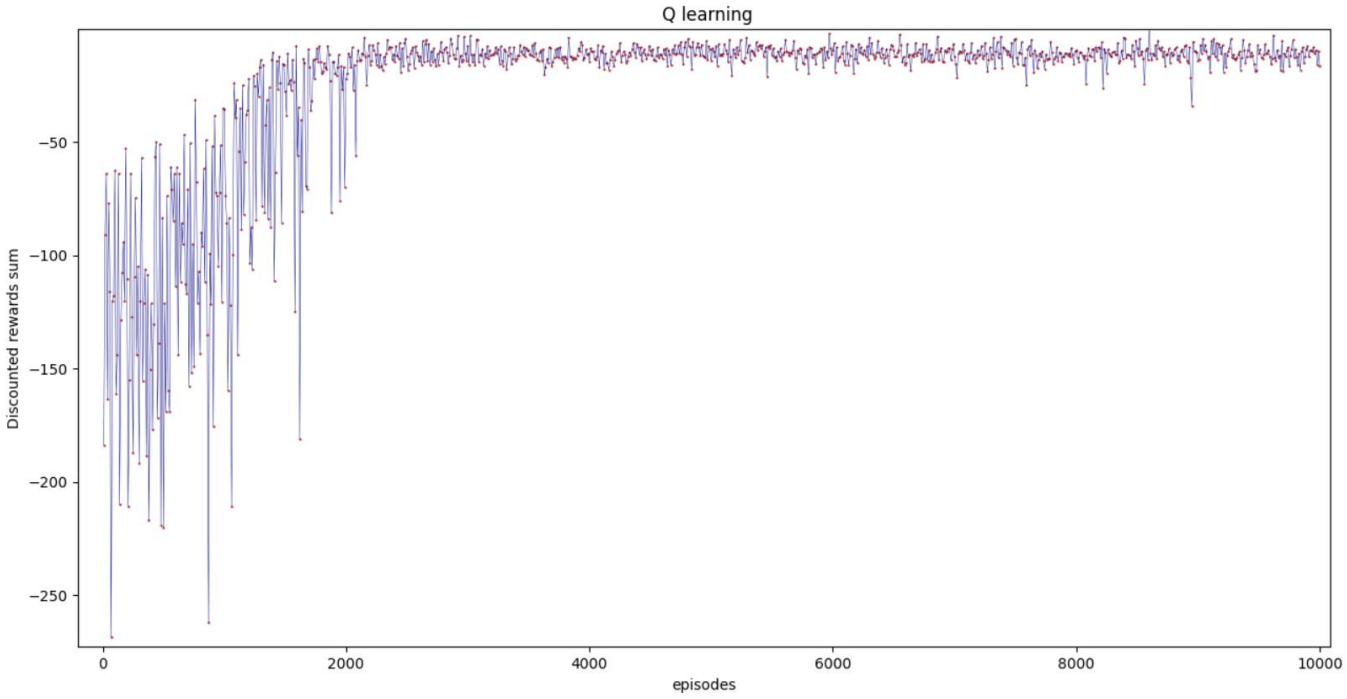
Inference -

With increase in alpha the ratio of correct drops to pickups increase. Showing the convergence is faster, learning is faster. Because the temporal difference value (from which learning happens) is weighted more. Whereas the maximum expected reward sum decreases.

5) Learning taxi for 10x10 maze.

Using Q learning with alpha = 0.3, discount = 0.99, epsilon = 0.1, 10000 episodes and maximum 1000 steps in each episode.

Graph is for destination Pink (9, 0).



Taxi	Passenger	destination	Minimum steps	Steps	Discounted reward sum
(1,4)	W (3,6)	P (9,0)	20	26	-6.66
(2,3)	M (6,5)	Y (0,1)	22	25	-5.72
(9,9)	Y (0,1)	C (8,9)	40	58	-32.33
(4,4)	R (0,9)	M (6,5)	25	34	-13.87
(0,0)	G (5,9)	W (3,6)	21	25	-5.72

Inference -

The number of steps taken is close to the minimum steps. And the difference is less when the steps required is less because wrong steps taken, random movement is lesser.

The discounted reward sum is negative because more than 20 steps are taken, all initial steps are -1 except last and the weight of reward decreases as the steps increase.