COL333 ASSIGNMENT 3

ARPIT CHAUHAN - 2019CS10332
AMAN GUPTA - 2019CS10673
November 26, 2021

- **PART A - Computing Policies**

1. Formulating the Taxi Domain as MDP

Any state in the *state space* S is described as a 5-tuple (x_taxi, y_taxi, x_psngr, y_psngr, flag) where flag = 1 when the passenger is inside the Taxi and 0 otherwise. For flag = 1, we will have a total of 25 states corresponding to each cell in the grid. For flag = 0, there will be 25X25 = 625 states corresponding to each combination of taxi and passenger positions. Thus including 1 terminal state, we obtain a total of 651 states in the MDP.

The *action space* A consists of 6 actions : (a) four navigation actions that move the taxi one grid cell to the North(N/0), South(S/1), East(E/2) of the West(W/3) directions, (b) a Pickup(PU/4) action, where the taxi attempts to pick up the passenger and (c) a Putdown(PD/5) action, where the taxi drops off the passenger. No further action is possible from the terminal state.

The *transition model*, T(s, a, s') = Prob(s'|s, a) is defined as per the following rules. Each of the four navigation actions succeeds with a probability of 0.85 and moves in a random other direction with a probability of 0.15 i.e. with a probability of 0.05 in each of the other three directions. Movements that attempt to cross the boundary of the grid or a wall result in no change in the state. The Pickup and Putdown actions are deterministic and lead to their exact intended effects with probability 1.

The *reward model,* R(s, a, s') is defined as per the following rules. 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 to the destination grid cell. Further, a reward of (-10) is received if the taxi attempts to *Pick Up* or *Put Down* a passenger when the taxi and the passenger are not located in the same grid cell.

R(0,0)				**G**(0,4)
		(2,2)		
Y(4,0)			**B**(4,3)	

2. Implementing Value Iteration

a) Value update equation for one ply of expectimax from each state is given by -

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

The implementation takes input an instance of the taxi domain, a parameter epsilon(denoting the maximum error allowed in the value of any state), and a discount factor γ (default $\gamma = 0.9$). We use max-norm distance in the successive value functions to determine convergence.

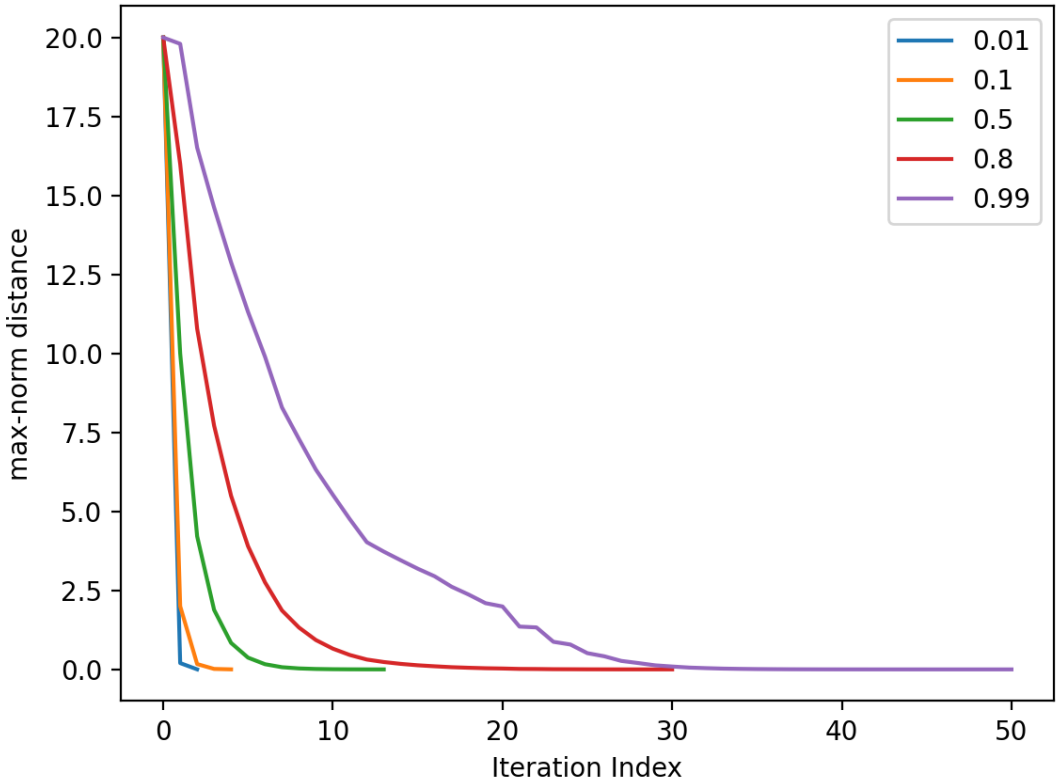
Value of epsilon chosen, $\epsilon = 0.001$

Number of iterations required for convergence, n = 39

b) In this part, we study the connection between the discount factor and the rate of convergence. We repeat part(a) by varying the discount factor in the range {0.01, 0.1, 0.5, 0.8, 0.99} and plot the max-norm distance vs. Iteration index for each one of them on the same graph. Value of epsilon is kept $\epsilon = 0.001$. The result obtained is attached below.
Number of iterations required for convergence = {3, 5, 14, 31, 51} respectively

Observations

As the discount factor increases, the number of iterations required for convergence increases, because when the discount factor is high, the agent attempts to maximize rewards for a longer action sequence, and hence the utility values of states keep changing for a larger number of iterations as rewards from far away states are propagated to it. For small discount factors, the agent is short-sighted, and hence the propagation of rewards does not take many iterations because the effect of reward at any particular state does not reach far-away states due to small γ.



c) In this part, we pick an instance: R(passenger initial location), B(passenger destination location), and (2,2) - Taxi initial location. Next, we simulate the policy obtained for the discount factor $\gamma = 0.1$ and $\gamma = 0.99$ by determining the first 20 states and actions prescribed by the policy.

Observations

We can observe that for $\gamma = 0.1$, the agent does not even pick the passenger in the first 20 steps, whereas for $\gamma = 0.99$, the agent picks up the passenger and drops him/her at the destination depot most of the time within 20 steps.
The actions chosen for $\gamma = 0.1$ do not seem to be optimal as the agent attempts to maximize immediate rewards which are not indicative of the final task that the agent needs to accomplish. The large reward available at the final destination does not get propagated sufficiently to the far-away states due to γ being very small.
The actions chosen for $\gamma = 0.99$ are optimal right from the start because it attempts to maximize reward over a long run and hence quickly moves to achieve the +20 reward reaching the final destination. Since the info of large rewards available at the destination gets propagated significantly to far-away states due to large γ, the policy becomes more goal-oriented and far-sighted.

Given below are the two state-action sequences obtained.

```
For discount factor gamma = 0.1
1   (2, 2, 0, 0, 0) N
2   (1, 2, 0, 0, 0) N
3   (0, 2, 0, 0, 0) N
4   (0, 2, 0, 0, 0) N
5   (0, 2, 0, 0, 0) N
6   (0, 2, 0, 0, 0) N
7   (0, 2, 0, 0, 0) N
8   (0, 2, 0, 0, 0) N
9   (0, 2, 0, 0, 0) N
10  (1, 2, 0, 0, 0) N
11  (1, 3, 0, 0, 0) N
12  (0, 3, 0, 0, 0) N
13  (0, 3, 0, 0, 0) N
14  (0, 3, 0, 0, 0) N
15  (0, 3, 0, 0, 0) N
16  (0, 2, 0, 0, 0) N
17  (0, 2, 0, 0, 0) N
18  (0, 2, 0, 0, 0) N
19  (0, 2, 0, 0, 0) N
20  (0, 2, 0, 0, 0) N
```

```
For discount factor gamma = 0.99
1   (2, 2, 0, 0, 0) W
2   (2, 1, 0, 0, 0) N
3   (1, 1, 0, 0, 0) N
4   (0, 1, 0, 0, 0) W
5   (0, 0, 0, 0, 0) PU
6   (0, 0, 0, 0, 1) S
7   (1, 0, 1, 0, 1) E
8   (1, 1, 1, 1, 1) S
9   (2, 1, 2, 1, 1) E
10  (2, 2, 2, 2, 1) E
11  (2, 3, 2, 3, 1) S
12  (3, 3, 3, 3, 1) S
13  (4, 3, 4, 3, 1) PD
14  (4, 3, 4, 3, -1) None
```

3. Implementing Policy Iteration

a) The policy evaluation step can be implemented in the following 2 ways -

Linear Algebra Method

Let V be the state value array of size 651, T be the transition matrix of dimension 651 X 651 where T[i][j] represents the probability of transition from state i to state j using action $\pi(i)$ as per the current policy π, R be the reward array of size 651 where for all non-terminal states, R[i] gives the immediate reward obtained on taking action $\pi(i)$ from state i and γ be the discount factor. An important observation here is that the reward is completely determined by the current state and action as per the policy, i.e. $R(s, \pi(s), s') = R(s, \pi(s))$ and hence R is 1-D instead of being 2-D. Now, the optimal values of states must follow the following matrix equation which represents |V| linear equations characterized by Bellman equations.

$$V = T(R + \gamma V)$$
$$V = TR + \gamma TV$$
$$(I - \gamma T)V = TR$$
$$V = (I - \gamma T)^{-1}TR$$

We use NumPy's linear algebra functions to solve the above relation and obtain the values of states.

Iterative Method

In this method, we start with V(s) = 0 for all states and then iterate using the following update rule until the values of states converge i.e. the max-norm distance between the value vectors obtained in successive iterations becomes less than epsilon (default $\epsilon = 0.001$).

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

Comparison

As the size of the state space S increases, the linear algebra method gets more and more computationally difficult as we need to calculate the inverse and perform matrix multiplication on large matrices of size up to |S|X|S| which takes $O(|S|^3)$ time and $O(|S|^2)$ space. The iterative method, although being computationally fast as it takes $O(|S|^2)$ time per iteration, only calculates approximate values as calculating the exact values would ideally take ∞ iterations. We stop the iterative method when the difference between consecutive max-norms becomes less than a certain threshold.

Hence, the Linear algebra method is preferred when the size of state space is small and the iterative method is more practical when the state space is large, due to the computational efficiency vs accuracy trade-off.

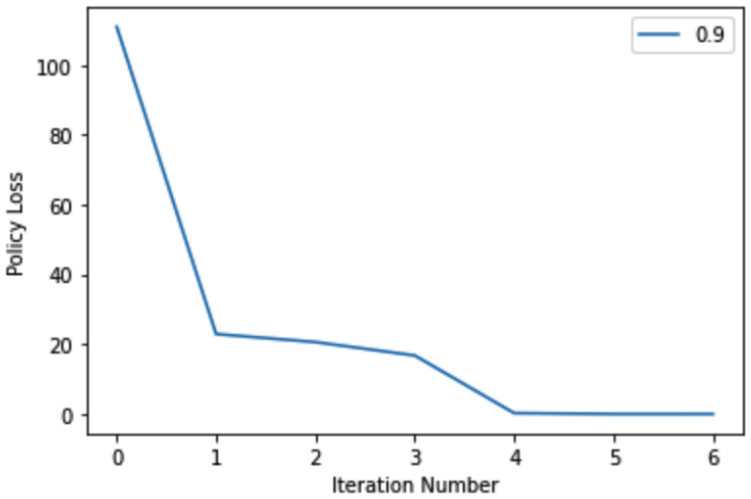
b) In this part, we run policy iteration till convergence to determine optimal policy using both the iterative method as well as the linear algebra method. Then, we repeat the process by computing the policy loss at each iteration between the current policy and the final optimal policy at convergence. In each iteration, after performing the policy evaluation using either of the two methods described in (a) part above, we use the following policy update rule -

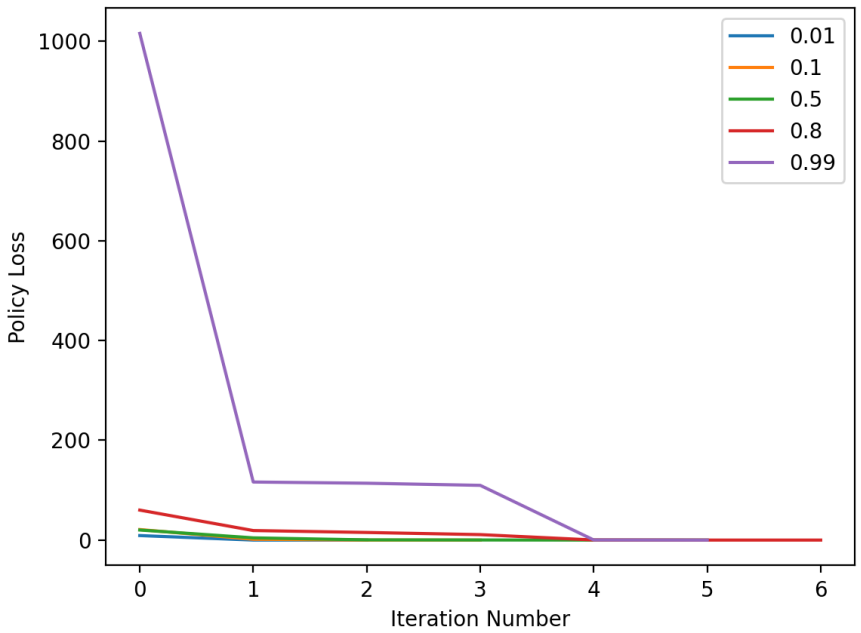
- **Policy Improvement:** For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s') \right]$$

We start with a randomly initialised policy. The policy iteration stops when the new improved policy comes out to be the same as the current policy.
The following plots were obtained for Policy loss vs Iteration number for varying values of the discount factor.

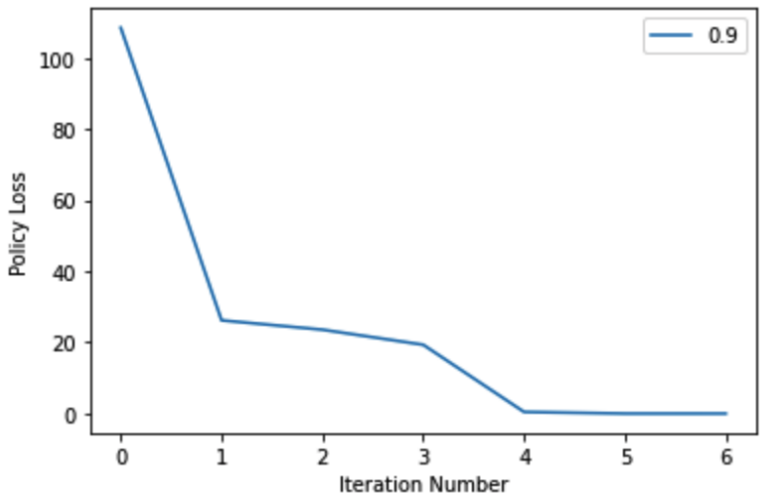
Iterative method

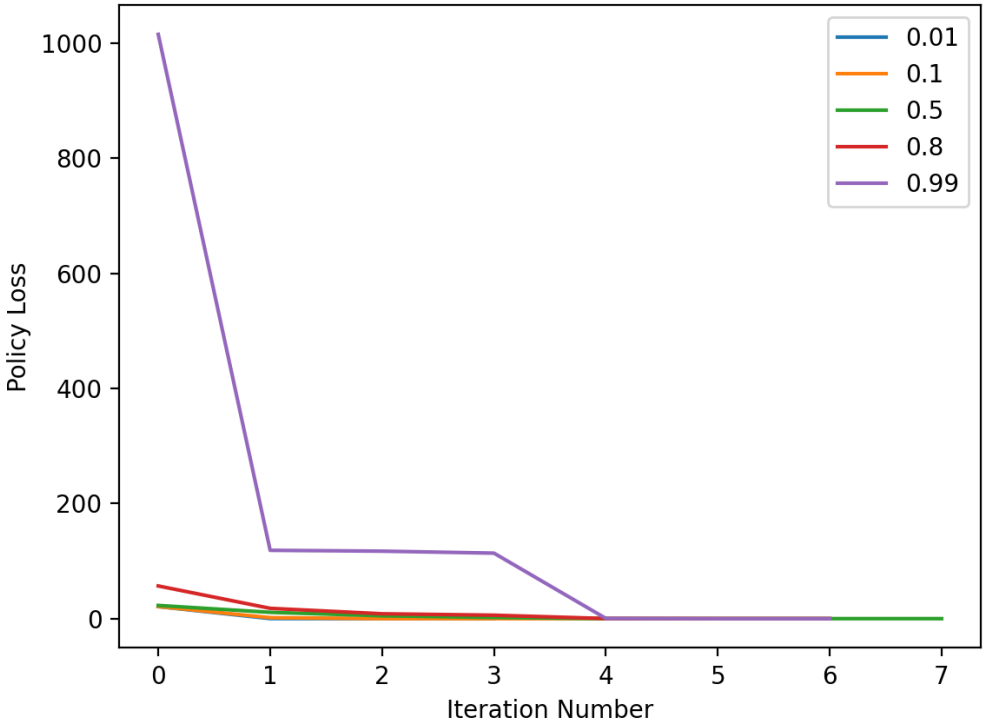




{3, 5, 6, 7, 7} iterations to converge for the discount factor in {0.01, 0.1, 0.5, 0.8, 0.99} respectively.

Linear Algebra Method





{6, 6, 7, 7, 7} iterations to converge for the discount factor in {0.01, 0.1, 0.5, 0.8, 0.99} respectively.

Observations

We observe that both the methods for policy evaluation yield almost similar graphs which is good because the policy should converge to the same values using either of the methods.

The number of iterations required to converge shows an increasing trend with increasing discount factor as a small discount factor is short-sighted and works on the basis of immediate rewards thus converging quickly whereas larger discount factors allow the final reward to propagate to far-away states thus taking more iterations.

Policy iteration takes a lesser number of iterations than Value iteration does because Policy is known to converge much before the values do.

Another observation is that the initial policy loss is very high for larger values of the discount factor. This is because, for a higher discount factor, we will be getting a policy closer to the optimal one which will be quite different from the initial random policy and hence higher initial policy loss.

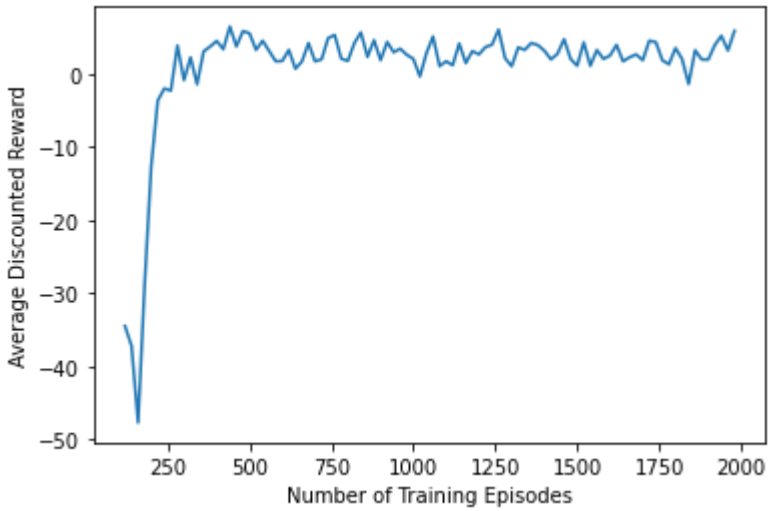
- **PART B - Incorporating Learning**

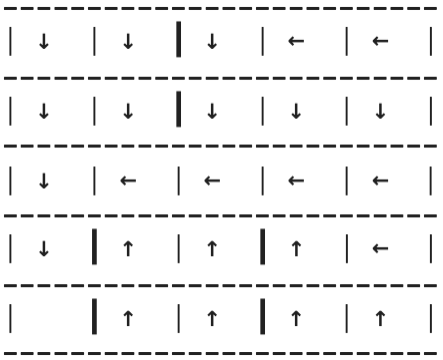
This part of the assignment involves obtaining optimal policy through reinforcement learning techniques. Unlike the last part we are not provided with the transition and reward model of the MDP. We are just given a simulation function of the environment which provides the immediate reward and next state on performing an action at a particular state.

1. a) This part performs Q-learning over 2000 episodes. Each episode starts with a random initial position of a taxi and a random depot apart from the destination depot for the person. In each step of the episode an action is chosen stochastically. With probability epsilon it is taken to be a random action and the action that maximizes q state utility with probability 1-epsilon. It is an on-policy algorithm. The equations governing the procedure are the following:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

With (small) probability ε, pick a random action
With (large) probability **1-**ε, act based on the current policy (based on the current Q –values in the table that the agent is updating)





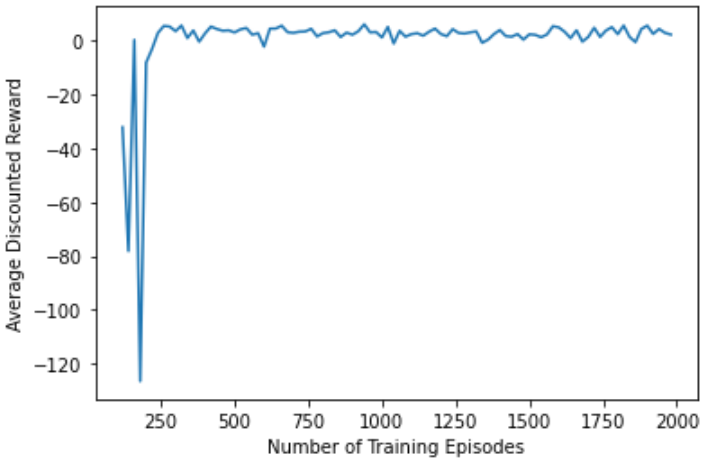
Optimal Policy when the person is inside the taxi

For the above graphs parameters are taken as :

Overall : 2000 training iterations
Epsilon: 0.1
Alpha: 0.25
Discount: 0.99

b) In this case Q learning with a decaying exploration rate is considered. The exploration rate decreases along with each learning step. The formula used is the following:

$$\epsilon = \frac{\epsilon}{LearningStep}$$



↓	↓	↓	↓	←
↓	↓	↓	↓	←
↓	←	←	←	↑
↓	↑	↑	↑	↑
	↑	→	↑	←

Using same parameters as before

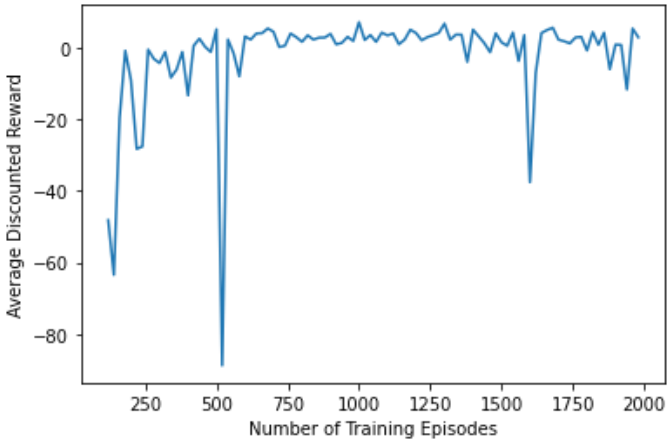
Observations:

- The discounted reward varies frequently in the initial phase of the graph as the epsilon value is relatively high in the beginning.
- The discounted reward converges better than in case of part (a) because with further training episodes the exploration rate decays to zero.

Note: Average over 10 episodes is considered for evaluation of policy at each stage. This was sufficient because the graphs did not have that high variance. This is done in all subparts of this type.

c) SARSA algorithm was used to obtain the optimal policy in this part. The main difference between SARSA and Q learning is that instead of taking the maximum of all possible actions for the lookahead state we consider a simulated reward and action for the same. It is an off-policy algorithm. It is governed by the following equation characterised by Q(s,a,r,s',r')

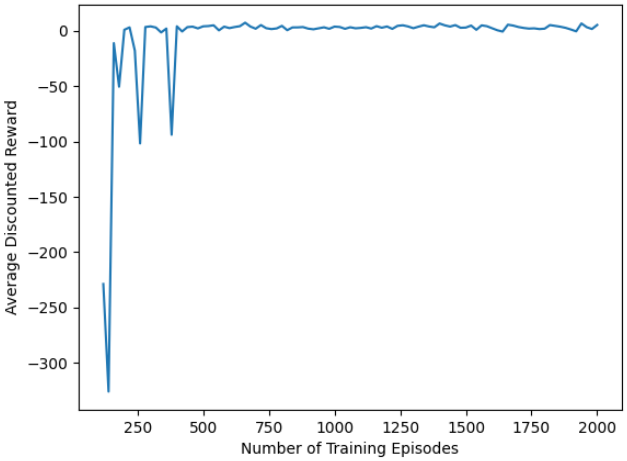
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma\, Q(s',a')\right]$$

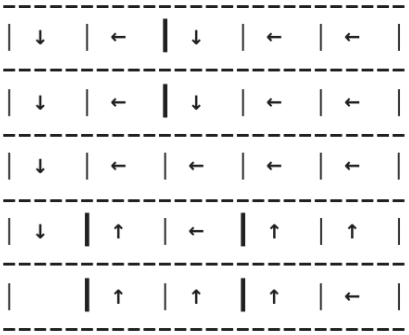


↓	↓	↓	←	←
↓	←	↓	←	↑
↓	←	←	←	←
↓	↑	↑	↑	↑
	↑	↑	↑	↑

d) In this part the SARSA was performed with a decaying exploration rate. The value of epsilon was varied inversely with the number of learning steps.

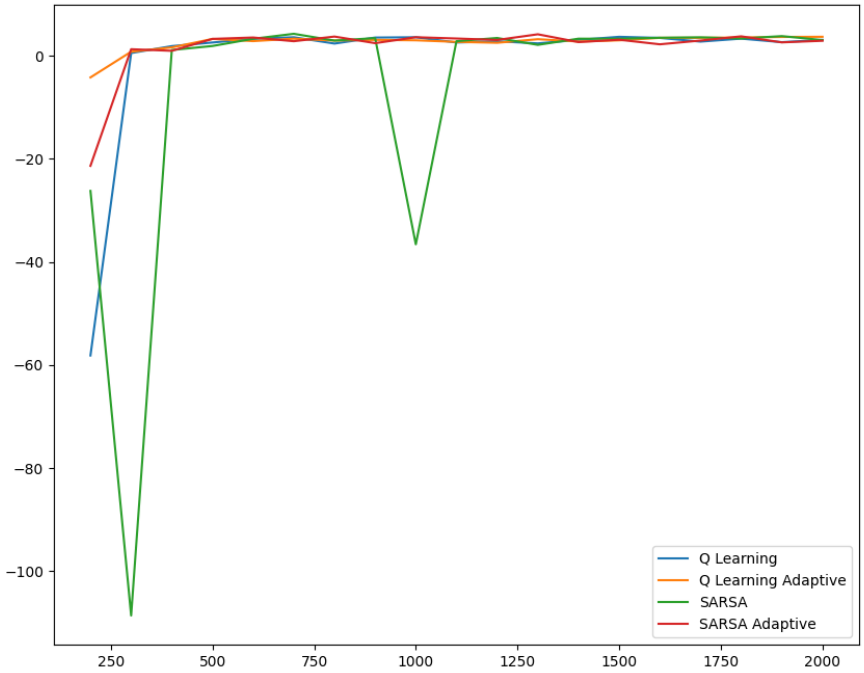
$$\epsilon = \frac{\epsilon}{LearningStep}$$





2. Comparison of Optimal Policies:

Optimal policies were obtained from the Q values obtained for each state-action pair in part1, by performing argmax of actions possibles on each state. Each algorithm was trained on exactly 2000 learning episodes. Then they were tested using the average discounted reward over exactly 100 random episodes. The learning rate (alpha) is set as 0.25 and discount factor (gamma) as 0.99.



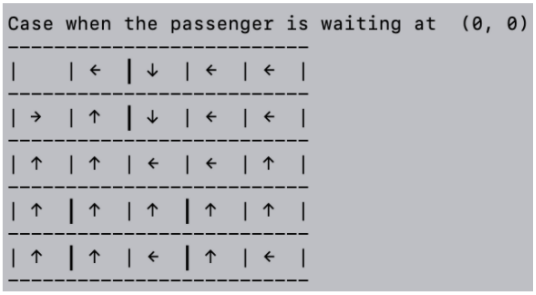
Algorithm	Reward
Q Learning	1.69
Q Learning Adaptive	2.59
SARSA	1.50
SARSA Adaptive	2.02

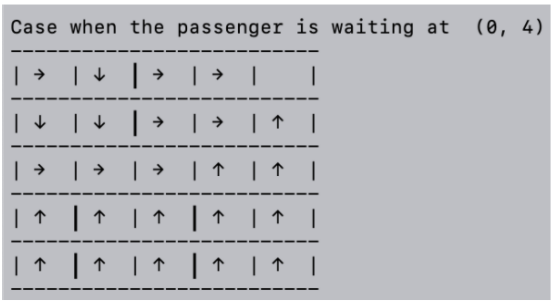
Observations

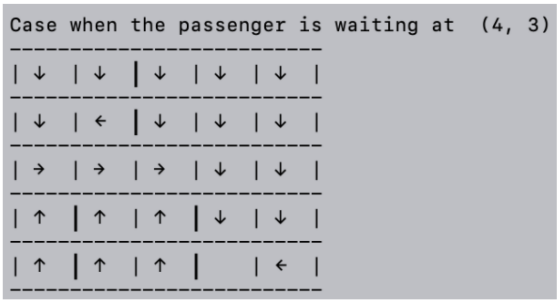
- The adaptive Q-Learning method works the best among the other algorithms as it converges to the high rewards in the fewest number of iterations. The average discounted rewards at the end were maximum for Q-learning with deteriorating exploration rate.
- SARSA doesn't converge as good as Q-Learning because there are some random episodes with very few rewards.
- Adaptive policies with decaying exploration rate worked better than the ones with constant epsilon because for them in the beginning episodes when we need to have exploration, epsilon is large and in the later episodes when we need to have exploitation, epsilon is low.

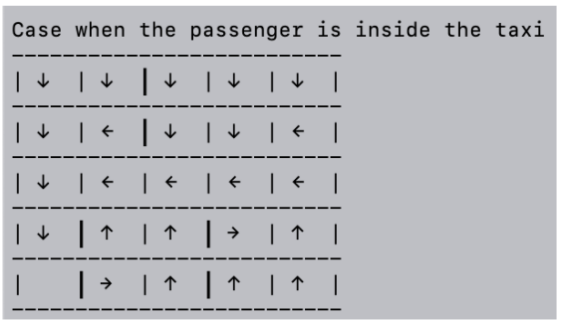
3. In this problem we simulate the policy obtained by Q Learning with a decaying exploration rate algorithm as it was the best in part 2. The policy is trained on the following parameters:
   Number of Learning Iterations: 10,000
   Discount: 0.99
   Alpha: 0.25
   Epsilon: 0.1(Initial)
   Destination Depot: Y(4,0)

The optimal policy is as plotted below:









The Simulation Obtained on 5 problem instances with varying initial depot of taxi and person are as follows:

```
Problem 1
Initial Taxi Depot  (0, 0)
Initial Person Depot  (0, 4)
1   (0, 0, 0, 4, 0) E
2   (0, 1, 0, 4, 0) S
3   (1, 1, 0, 4, 0) S
4   (2, 1, 0, 4, 0) E
5   (2, 2, 0, 4, 0) E
6   (2, 3, 0, 4, 0) N
7   (3, 3, 0, 4, 0) N
8   (2, 3, 0, 4, 0) N
9   (2, 2, 0, 4, 0) E
10  (2, 3, 0, 4, 0) N
11  (1, 3, 0, 4, 0) E
12  (1, 4, 0, 4, 0) N
13  (1, 4, 0, 4, 0) N
14  (1, 3, 0, 4, 0) E
15  (1, 4, 0, 4, 0) N
16  (0, 4, 0, 4, 0) PU
17  (0, 4, 0, 4, 1) S
18  (0, 4, 0, 4, 1) S
19  (1, 4, 1, 4, 1) W
20  (1, 4, 1, 4, 1) W
21  (1, 3, 1, 3, 1) S
22  (2, 3, 2, 3, 1) W
23  (1, 3, 1, 3, 1) S
24  (2, 3, 2, 3, 1) W
25  (2, 2, 2, 2, 1) W
26  (2, 1, 2, 1, 1) W
27  (1, 1, 1, 1, 1) W
28  (1, 0, 1, 0, 1) S
29  (2, 0, 2, 0, 1) S
30  (3, 0, 3, 0, 1) S
31  (4, 0, 4, 0, 1) PD
32  (4, 0, 4, 0, -1) None
Reward  -11.235955193406339
```

```
Problem 2
Initial Taxi Depot  (0, 4)
Initial Person Depot  (4, 3)
1   (0, 4, 4, 3, 0) S
2   (1, 4, 4, 3, 0) S
3   (2, 4, 4, 3, 0) S
4   (3, 4, 4, 3, 0) S
5   (4, 4, 4, 3, 0) W
6   (4, 4, 4, 3, 0) W
7   (4, 3, 4, 3, 0) PU
8   (4, 3, 4, 3, 1) N
9   (3, 3, 3, 3, 1) E
10  (3, 4, 3, 4, 1) N
11  (2, 4, 2, 4, 1) W
12  (3, 4, 3, 4, 1) N
13  (2, 4, 2, 4, 1) W
14  (2, 3, 2, 3, 1) W
15  (2, 2, 2, 2, 1) W
16  (2, 1, 2, 1, 1) W
17  (2, 0, 2, 0, 1) S
18  (3, 0, 3, 0, 1) S
19  (4, 0, 4, 0, 1) PD
20  (4, 0, 4, 0, -1) None
Reward  0.14165137401051453
```

```
Problem 3
Initial Taxi Depot  (4, 3)
Initial Person Depot  (0, 4)
1   (4, 3, 0, 4, 0) N
2   (3, 3, 0, 4, 0) N
3   (2, 3, 0, 4, 0) N
4   (1, 3, 0, 4, 0) E
5   (2, 3, 0, 4, 0) N
6   (1, 3, 0, 4, 0) E
7   (1, 4, 0, 4, 0) N
8   (0, 4, 0, 4, 0) PU
9   (0, 4, 0, 4, 1) S
10  (1, 4, 1, 4, 1) W
11  (1, 3, 1, 3, 1) S
12  (2, 3, 2, 3, 1) W
13  (2, 2, 2, 2, 1) W
14  (2, 1, 2, 1, 1) W
15  (2, 0, 2, 0, 1) S
16  (1, 0, 1, 0, 1) S
17  (2, 0, 2, 0, 1) S
18  (1, 0, 1, 0, 1) S
19  (1, 0, 1, 0, 1) S
20  (2, 0, 2, 0, 1) S
21  (3, 0, 3, 0, 1) S
22  (4, 0, 4, 0, 1) PD
23  (4, 0, 4, 0, -1) None
Reward  -2.8326558134489694
```

```
Problem 4
Initial Taxi Depot  (4, 3)
Initial Person Depot  (0, 0)
1   (4, 3, 0, 0, 0) N
2   (3, 3, 0, 0, 0) N
3   (2, 3, 0, 0, 0) W
4   (2, 2, 0, 0, 0) W
5   (2, 1, 0, 0, 0) N
6   (1, 1, 0, 0, 0) N
7   (0, 1, 0, 0, 0) W
8   (0, 0, 0, 0, 0) PU
9   (0, 0, 0, 0, 1) S
10  (1, 0, 1, 0, 1) S
11  (2, 0, 2, 0, 1) S
12  (3, 0, 3, 0, 1) S
13  (4, 0, 4, 0, 1) PD
14  (4, 0, 4, 0, -1) None
Reward  6.366184605935512
```

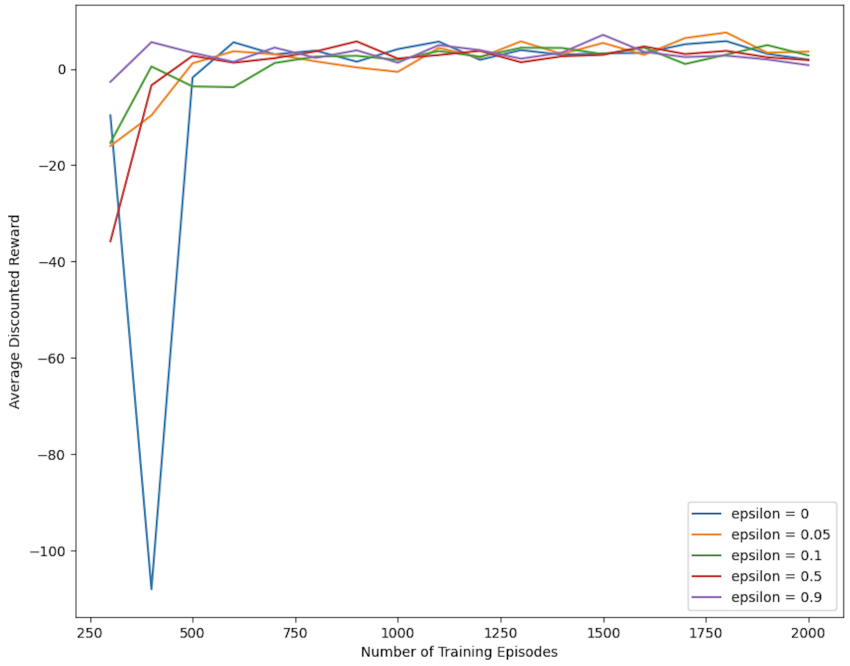
In all the cases the taxi collects the passenger at their location depot and drops them at the destination depot
Average Reward = **-0.87**

```
Problem 5
Initial Taxi Depot  (0, 4)
Initial Person Depot  (0, 0)
1   (0, 4, 0, 0, 0) W
2   (0, 3, 0, 0, 0) W
3   (0, 2, 0, 0, 0) S
4   (1, 2, 0, 0, 0) S
5   (2, 2, 0, 0, 0) W
6   (2, 1, 0, 0, 0) N
7   (1, 1, 0, 0, 0) N
8   (2, 1, 0, 0, 0) N
9   (1, 1, 0, 0, 0) N
10  (0, 1, 0, 0, 0) W
11  (0, 0, 0, 0, 0) PU
12  (0, 0, 0, 0, 1) S
13  (1, 0, 1, 0, 1) S
14  (2, 0, 2, 0, 1) S
15  (3, 0, 3, 0, 1) S
16  (4, 0, 4, 0, 1) PD
17  (4, 0, 4, 0, -1) None
Reward  3.207002556954622
```

Analysis:

- With all possible initial depots of the person and taxi, following the optimal policy enables the taxi to collect the person from their depot and drop them at the destination depot.
- The optimal policy is only dependent on the destination depot and not on the initial depot of taxi or person.

4. In this part we analyse the effect of learning rate alpha and epsilon on the Q-Learning Algorithm. First we keep the learning rate alpha as 0.1 and vary the exploration rate epsilon value in the range {0,0.01,0.1,0.5,0.9}. Other parameters were; gamma = 0.99, alpha = 0.1, number of learning episodes = [0,2000,20], 10 episodes were considered to evaluate the policy. The following graph was obtained:

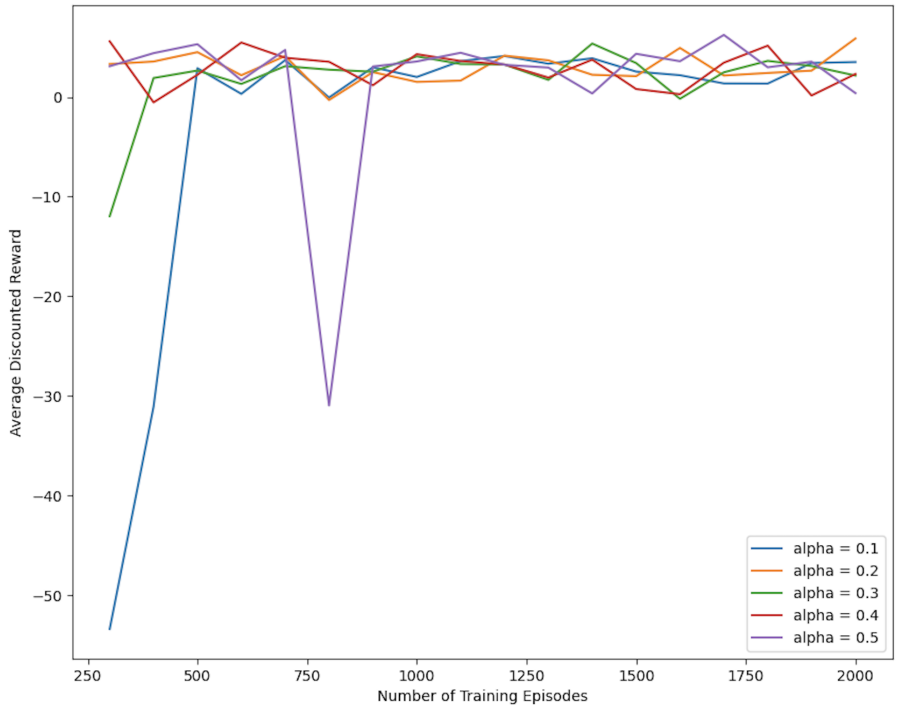


Observation and Inference

The main role of parameter epsilon is to introduce **randomness** in the search. With the aim to explore new states instead of just repeating the path along the same policy which may be wrong at some states, it forces the algorithm to deliberately look for new states so that the policy is correct at most of the states in the space.

- We notice that the policy with epsilon equal to 0.9 gives the best reward in the beginning of the graph, this is because it mainly focuses on exploring new states, which should be the main motive in the initial episodes. But as the number of episodes proceed we need to follow the optimal path as the exploitation is the main motive in those. As the epsilon is quite high (0.9) even in the later stage it does not converge and has a high variance. This is the reason why decaying exploration rate works better than constant rate.
- For the low epsilon range from 0 to 0.1, the rewards improve with the increase in epsilon as the exploration factor is causing discovery of new states. This helps in convergence of policy. We notice that policy with zero epsilon also converges; this occurs because in those cases randomness is still present due to the presence of stochastic actions.

Next to observe the effect of learning rate alpha we fix the epsilon as 0.1 and vary alpha in range {0.1,0.2,0.3,0.4,0.5}. Rest of the parameters are the same. The following graph was observed:



Observation: As the learning rate increases the rate of convergence of policies also increases as the utility at each state is updated at a faster rate. This is the reason that alpha equals 0.5 has the highest reward in the beginning. There is also a negative impact for a higher learning rate. Suppose that a stochastically less probable action occurs at a state, ideally this should not affect the utility much but because of high learning rate its weightage is increased, therefore we observe random crests of low reward with high learning rate.

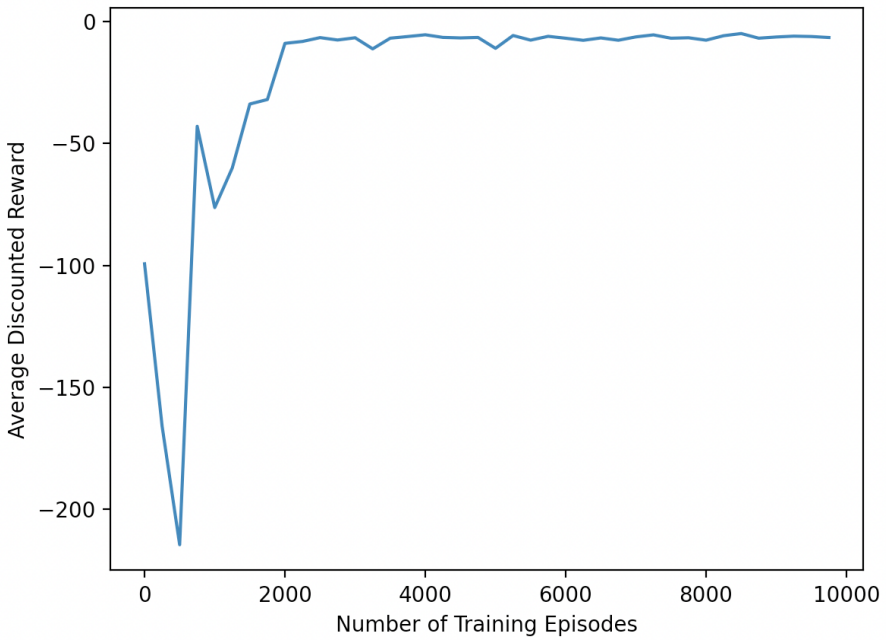
5. In this part we apply the RL strategies to a bigger grid of 10X10 size. The method followed was Q-Learning with decaying exploration rate. A total of 10000 learning iterations were performed. The problem was tested with 8 different instances each with a different destination depot. The parameters taken were as follows:

   Learning Rate Alpha: 0.5
   Discount: 0.99
   Initial Epsilon: 0.1

   The testing was done using random episodes with starting depot as one of the depots other than the destination depot and starting location of Taxi as any random location on the graph. The discounted reward obtained on the 8 instances are as follows:

Destination Depot	Initial Depot	Starting Taxi Pos.	Discounted Reward
0,0	9,4	4,5	-2.83
0,5	3,3	2,9	-6.66
3,3	0,5	3,9	-5.71
9,4	3,3	0,4	6.37
4,6	3,3	0,4	-1.85
9,9	0,0	1,5	-17.27
0,8	0,0	4,2	-11.26
8,0	0,5	2,4	-0.85

Average Discounted Reward over these 8 iterations is -4.



Observations:

- As can be observed from the graph that the policy is very random in the beginning leading to large negative responses. In the Adaptive Q learning strategy the exploration keeps on decreasing across episodes therefore as the learning episodes increases the exploration is converted to exploitation.
- Since it's a large grid and almost every location has a -1 reward except reaching the terminal state, which has +20 reward. The taxi takes a longer time to drop the person to the destination position hence we notice negative rewards even with the converged policy.

END