COMP532 ASSIGNMENT 1

REINFORCEMENT LEARNING

SUBMITTED BY

USMAN SHOUKAT	STUDENT ID: 201537600
JONE CHONG	STUDENT ID: 201533109
SAHIB BIR SINGH BHATIA	STUDENT ID: 201547831
PARTH KHANDELWAL	STUDENT ID: 201549274

INDEX

TABLE OF FIGURES

Q1 Re-implement in Python the results presented in Figure 2.2of the Sutton & Barto book comparing a greedy method with two ε-greedy methods (*ε*=0.01and *ε*=0.1), on the 10-armedtestbed, and present your code and results. Include a discussion of the exploration -exploitation dilemma in relation to your findings.

Answer:

Reinforcement learning is a form of machine learning in which an agent learns from its interaction with an environment to achieve a given goal.

A distinctive challenge that arises in reinforcement learning lies in balancing between exploration and exploitation.

Exploitation means that the agent selects the action that has the highest average rewards. Whereas**, Exploration** means that the agent selects an action randomly, regardless of the reward values obtained previously.

A simple method for performing the balancing between exploration and exploitation is the method called ε- greedy. This method behaves most of the time greedily, yet now and then, with a small probability ε, randomly selects one action among all actions

We compare the two greedy methods with two different static ***ε*=0.01**and ***ε*=0.1** on the 10 armed test bed over 2000 iterations**.** After running the code we got the following results.

In the two graphs shown below in figure 1, We observed that:

- The **greedy method has the lowest average reward and optimal action %.** This demonstrates pure exploitation has worse results in the long run due to its inhibition of finding the optimal action (as compared to ε-greedy methods).

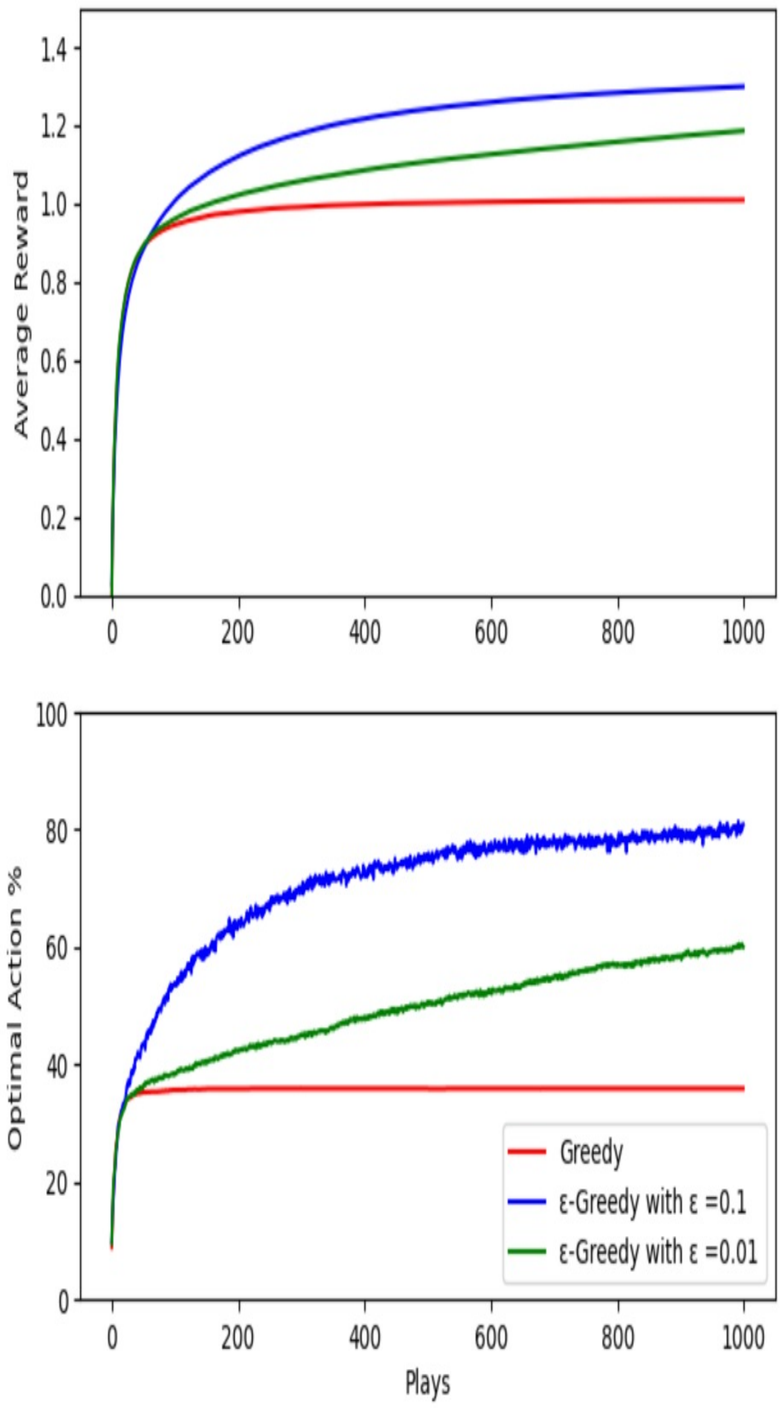


Figure 1

- **For the ε-greedy method of 0.01**, exploration is introduced into decision making. This results in the increase of optimal action % but at a slower rate than 0.1. However, judging from the results: **over more iterations, the 0.01 will eventually surpass the 0.1 method in optimal action % and average reward.**

- **For the ε-greedy method of 0.1**, exploration is implemented to a higher degree, leading to it finding the optimal action faster, but levels off too early compared to 0.01.

The comparison of the ε-greedy method of 0.1 and 0.01 shows that a higher ε-greedy value (more exploration) results in a quicker improvement but would plateau in optimal action % average reward.

In relation to the exploration-exploitation dilemma, an updated balance of exploitation and exploration can be implemented in the form of a decaying ε-greedy value. By having a relatively higher ε-greedy value at the start, then reducing the ε-greedy value over time, it would result in a faster improvement due to high exploration, and increased optimal action %; then, having more exploitation at the end, capitalizing on the exploration of action value estimates from the early stages.

Q2 Consider a Markov Decision Process (MDP)with states S={4,3,2,1,0}, where 4 is the starting state. In states k≥1you can walk (W) and $T(k,W,k-1)=1$. In states k≥2you can also jump (J) and $T(k,J,k-2)=3/4$and $T(k,J,k)=1/4$. State 0 is a terminal state. The reward $R(s,a,s')=(s-s')2$for all (s,a,s'). Use a discount of $\gamma=1/2$. Compute both $V*(2)$and $Q*(3,J)$. Clearly show how you computed these values.

Answer:

- **Finding V*(2)**

  The value function can be defined as follows:-

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

  The given values are as follows:
  - States = {4,3,2,1,0}
  - γ=1⁄2
  - Reward R(s, a, s') = (s-s')^2
  - T(k,j,k-2) = ¾
  - T(k, J, k) = ¼
  - State 0 is the terminal state
  - State 4 is the starting state

  Now, putting values in the formula above,

  V*(2) = max[((s-s')^2+ γV*(1), T(2,J,0)[(s-s')^2 + γV*(0)] +T(2,J,2)[(s-s')^2 + γV*(2)]]

  In order to compute V*(2) we need to first calculate V*(0) and V*(1).

  V*(0) = 0 [as given in the question as it is a terminal state]

  V*(1) = max[(s-s')^2 + γV*(0)]

  V*(1) = max [(0-1)^2 + ½ * 0] = max[1] = 1

  Now, putting value in the above equation:-

V*(2) = max[(0-1)^2 + ½*1, 3/4[(0-2)^2 + ½(0)] + ¼[(2-2)^2 + 1/2*V(2)]]

V*(2) = max([1 + 0.5 , ¾(4 + 0) + ¼(0 +1/2V*(2)]

V*(2) = max[1.5,3+ 1/8V*(2)]

Now, considering Jump action as the optimal action we get:-

V*(2) = 3 + 1/8 V*(2)

V*(2) - 1/8 V*(2) = 3

V*(2) = 24/7 = 3.428

Putting this value in equation 1, we get

V*(2) = max[1.5, 3.428]

Therefore,

$$\boxed{\mathbf{V^*(2) = 3.428}}$$

- **Finding Q*(3,J)**

The action-value function can be defined as follows: -

$$Q^*(s,a) = \sum_{s'} \mathbf{P}^a_{ss'}\left[\mathbf{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right]$$

Now, putting values in the formula above,

Considering the optimal action at state 3 is Jumping

Q*(3,J) = ¾[(3-1)^2 + ½(Q*(1,W))] + ¼[(3-3)^2 + ½(Q*(3,J))]

In order to compute Q*(3,J) we need to first calculate Q*(1,W)

Q*(1,W) = 1[(1-0)^2 + ½(Q*(0,0))]

We have Q*(0,0) = 0, then

Q*(1,W) = 1 +1/2(0) = 1

Then, putting above value of Q*(1,W) to calculate Q*(3,J) we have

Q*(3,J) = ¾[4+ ½(1)] + ¼[0 + ½(Q*(3,J))]

Q*(3,J) = ¾(9/2) + 1/8Q*(3,J)

7/8Q*(3,J) = 27/8

Q*(3,J) = 27/7 = 3.857

Now, considering the optimal action at state 3 is walking

Q*(3,J) = ¾[(3-1)^2 + ½(Q*(1,W))] + ¼[(3-3)^2 + ½(Q*(3,W))]

In order to compute Q*(3,J) we need to first calculate Q*(3,W):

Q*(3,W) = 1 +1/2 Q*(2,J)

Q*(3,W) = 1 +1/2 (24/7) = 19/7

Then, putting above value of Q*(3,W) to calculate Q*(3,J) we have:

Q*(3,J) = ¾[4+ ½(1)] + ¼[0 + ½(Q*(3,W))]

Q*(3,J) = ¾(9/2) + 1/8(19/7)

Q*(3,J) = 27/8 + 19/56

Q*(3,J) = 3.71

After comparison we can say the optimal action was Jumping so the final Q*(3,J) is as follows:

$$\boxed{\mathbf{Q^*(3,J) = 3.857}}$$

Q3 Consider the following Reinforcement Learning problem (the rewards R are tagged to the transitions, the transition probabilities are unknown) with states 1...7, of which state 7 is a terminal state. Let the initial values of all states be 0. Initialize the discount factor $\gamma = 1$. What are the values of all states (after each epoch) when Temporal Difference learning is used after the following episodes? The learning parameter $\alpha = 0.5$ is fixed.

Episode 1: {1, 3, 5, 4, 2, 7}

Episode 2: {2, 3, 5, 6, 4, 7}

Episode 3: {5, 4, 2, 7}

Answer: The state value can be computed by using the below formula:-

$$V(s) = V(s) + \alpha[r + \gamma*V(s+1) – V(s)]$$

- **Episode 1 {1,3,5,4,2,7}**
  By applying the above equation in episode one we get the following equations:
  $V(1) = V(1) + \alpha[r + \gamma*V(3) – V(1)]$ – i
  $V(3) = V(3) + \alpha[r + \gamma*V(5) – V(3)]$ – ii
  $V(5) = V(5) + \alpha[r + \gamma*V(4) – V(5)]$ - iii
  $V(4) = V(4) + \alpha[r + \gamma*V(2) – V(7)]$ – iv
  $V(2) = V(2) + \alpha[r + \gamma*V(7) – V(2)]$ – v

  From **eq(i)** we can calculate V(1) with initial value of it as 0 and $V(3) = 0$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  $V(1) = 0 + 0.5[-2 + 1(0) – 0]$ => $\mathbf{V(1) = -1}$

  From **eq(ii**) we can calculate V(3) with initial value of it as 0 and $V(5) = 0$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  $V(3) = 0 + 0.5[4 + 1(0) – 0]$ => $\mathbf{V(3) = 2}$

From **eq(iii**) we can calculate V(5) with initial value of it as 0 and V(4) = 0 and $\gamma = 1$ and $\alpha = 0.5$ as follows:

V(5) = 0 + 0.5[2 + 1(0) – 0] => **V(5) = 1**

From **eq(iv**) we can calculate V(4) with initial value of it as 0 and V(2) = 0 and $\gamma = 1$ and $\alpha = 0.5$ as follows:

V(4) = 0 + 0.5[-2 + 1(0) – 0] => **V(4) = -1**

From **eq(v**) we can calculate V(2) with initial value of it as 0 and V(7) = 0 and $\gamma = 1$ and $\alpha = 0.5$ as follows:

V(2) = 0 + 0.5[4 + 1(0) – 0] => **V(2) = 2**

- **Episode 2 {2,3,5,6,4,7}**
  By applying the equation in episode two we get the following equations:
  $V(2) = V(2) + \alpha[r + \gamma*V(3) – V(2)]$ – i
  $V(3) = V(3) + \alpha[r + \gamma*V(5) – V(3)]$ – ii
  $V(5) = V(5) + \alpha[r + \gamma*V(6) – V(5)]$ - iii
  $V(6) = V(6) + \alpha[r + \gamma*V(4) – V(6)]$ – iv
  $V(4) = V(2) + \alpha[r + \gamma*V(7) – V(4)]$ – v

  From **eq(i)** we can calculate V(2) = 2 and V(3) = 2 and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  V(2) = 2 + 0.5[-2 + 1(2) – 2] => **V(2) = 1**

  From **eq(ii)** we can calculate V(3) = 2 and V(5) = 1 and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  V(3) = 2 + 0.5[4 + 1(1) – 2] => **V(3) = 3.5**

  From **eq(iii)** we can calculate V(5) = 1 and V(6) = 0 and $\gamma = 1$ and $\alpha = 0.5$ as follows:

$V(5) = 1 + 0.5[3 + 1(0) – 1]$ => $\mathbf{V(5) = 2}$

From **eq(iv)** we can calculate $V(6) = 0$ and $V(4) = -1$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

$V(6) = 0 + 0.5[1 + 1(-1) – 0]$ => $\mathbf{V(6) = 0}$

From **eq(v)** we can calculate $V(4) = -1$ and $V(7) = 0$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

$V(4) = -1 + 0.5[-1 + 1(0) +1]$ => $\mathbf{V(4) = -1}$

- **Episode 3 {5,4,2,7}**
  By applying the equation in episode two we get the following equations:
  $V(5) = V(5) + \alpha[r + \gamma*V(4) – V(5)]$ – i
  $V(4) = V(4) + \alpha[r + \gamma*V(2) – V(4)]$ – ii
  $V(2) = V(2) + \alpha[r + \gamma*V(7) – V(2)]$ – iii

  From **eq(i)** we can calculate $V(5) = 2$ and $V(4) = -1$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  $V(5) = 2 + 0.5[2 + 1(-1) - 2]$ => $\mathbf{V(5) = 1.5}$

  From **eq(ii)** we can calculate $V(4) = -1$ and $V(2) = 1$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  $V(4) = -1 + 0.5[-2 + 1(1) +1]$ => $\mathbf{V(4) = -1}$

  From **eq(iii)** we can calculate $V(2) = 1$ and $V(7) = 0$ and $\gamma = 1$ and $\alpha = 0.5$ as follows:

  $V(2) = 1 + 0.5[4 + 1(0) -1]$ => $\mathbf{V(2) = 2.5}$

The following table represents the state values after each episode

Episode	**V(1)**	**V(2)**	**V(3)**	**V(4)**	**V(5)**	**V(6)**	**V(7)**
1	-1	2	2	-1	1	0	0
2	-1	1	3.5	-1	2	0	0
3	-1	2.5	3.5	-1	1.5	0	0

Q4

a)What does the Q-learning update rule look like in the case of a stateless or 1-state problem? Clarify your answer.

Answer:

The Q- learning algoirthm is one of the most often used algoirthms in reinforcement learning.[1]This algorithm computes the table of Q (s, a) values (called Q-table) which represent the expected reward that an agent can obtain in a state s after it performs an action a. The Q-table is updated by successive approximations for an actual state-action pair (st,at) according to the following formula

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \\ + \alpha\left(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right),$$

where the maximization operator refers to the action value a which may be performed in next the state st+1 and α ∈ (0, 1] is the learning rate.

In the case when it is impossible to distinguish the state of the system, or the system is static remaining solely in one state, the reinforcement learning algorithm takes the reduced stateless form also called the single-state Q-learning. The rule of the action value function update is then the following: Qt+1 (at) = Qt (at) + α (rt − Qt (at)) which after simple calculations, can be express as follows

$$Q_{t+1}(a_t) = Q_t(a_t)(1 - \alpha) + \alpha r_t.$$

The Q-table is therefore reduced to the form of a vector.[2]

b)Discuss the main challenges that arise when moving from single- to multi-agent learning, in terms of the learning target and convergence.

Answer:

- The multi-agent framework is based on the same idea of single agent learning but,there are several agents deciding on actions over the environment. The big difference resides in the fact that all each agent probably has some effect on the environment and, so, actions can have different outcomes depending on what the other agents are doing. [3]

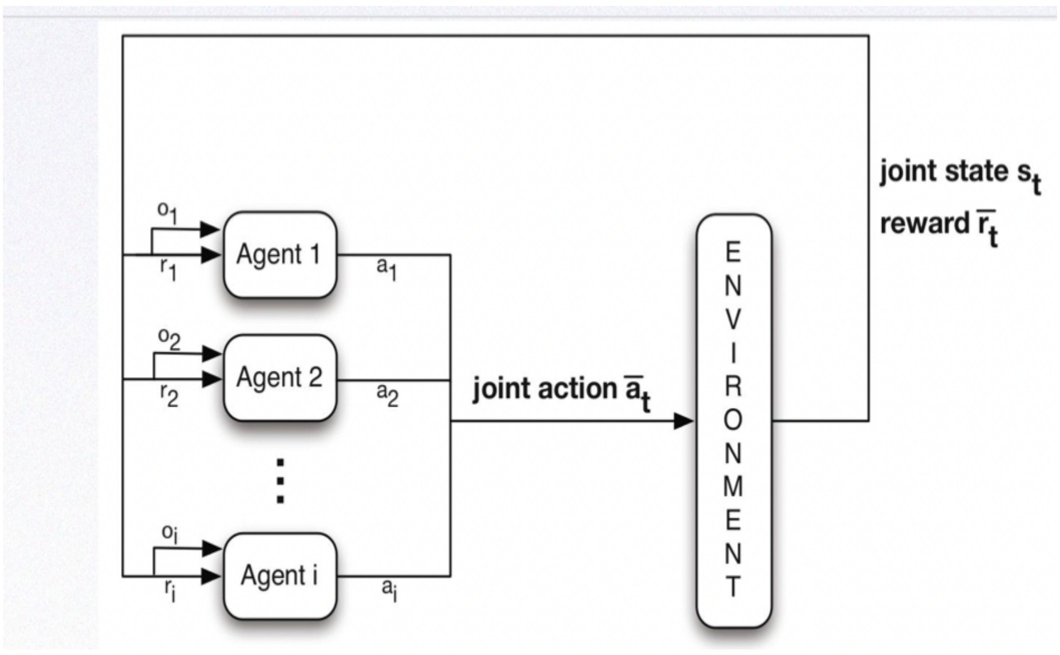


Figure 2 Multi Agent Learning Framework

- In multi-agent learning, multiple agents that make decisions affect the actions taken by all other agents, with all of them attempting to maximize their reward. **Learning in multi agent systems poses the the problem of non stationary or non convergence due to interactions between other agents.**

- Specifying a good MARL goal in the general stochastic game[redefined version of MDP for Multi agent learning] is a difficult challenge, because the agents' returns are correlated and cannot be maximized independently.

- Another challenge is the exploration-exploitation trade-off which requires online (single-agent as well as multi-agent) RL algorithms to strike a balance between the exploitation of the agent's current knowledge, and exploratory, information-gathering actions taken to improve that knowledge. For instance, the Boltzmann equation is a simple way of trading off exploration with exploitation. The exploration procedure is crucial for the efficiency of RL algorithms.

- In Multi agent learning, some complications arise due to the presence of multiple agents. Agents explore to obtain information not only about the environment, but also about the other agents in order to adapt to their behavior. Too much exploration, however, can destabilize the other agents, thereby making the learning task more difficult for the exploring agent.

- Generally, in an MDP, there exists at least one optimal policy that is stationary and deterministic, but due to the multiple agents, there is no optimal policy as there is a dependence on the behaviour of other agents. This results in a difficulty of converging to a Nash-equilibrium.[4]

Q5. Re-implement in Python the results presented in Figure 6.4 of the Sutton & Barto book comparing SARSA and Q-learning in the cliff-walking task. Investigate the effect of choosing different values for the exploration parameter e for both methods. Present your code and results. In your discussion clearly describe the main difference between SARSA and Q-learning in relation to your findings.

Note: For this problem, use α = 0.1 and γ = 1 for both algorithms. The "smoothing" that is mentioned in the caption of Figure 6.4 is a result of 1) averaging over 10 runs, and 2) plotting a moving average over the last 10 episodes.

Answer:

SARSA is a on policy algorithm which means that it learns action values relative to the policy it follows.

$Q(s_{t+1},a_{t+1}) = \varepsilon \cdot \text{mean}_a Q(s_{t+1},a) + (1-\varepsilon) \cdot \max_a Q(s_{t+1},a)$
In SARSA both current state estimation as well as next state estimation is done using ε-Greedy approach.

Q Learning is an off-policy algorithm that learns action values using an e-greedy approach for the current state, and next state estimation is done using the normal greedy approach.

In practical terms, under the ε-greedy policy, Q-Learning computes the difference between Q(s,a) and the maximum action value, while SARSA computes the difference between Q(s, a) and the weighted sum of the average action value and the maximum:

$Q(s_{t+1},a_{t+1}) = \max_a Q(s_{t+1},a)$

In the problem statement, we were asked to use an alpha value of 0.1, but using this value, we had a different graph from the graph given in the book. And when we consulted it with the book, we learned that the graph in the book is plotted with an alpha value of

0.5. So, we carried out our analysis with an alpha value of 0.5. We are also attaching the figure we were getting with an alpha value of 0.1.

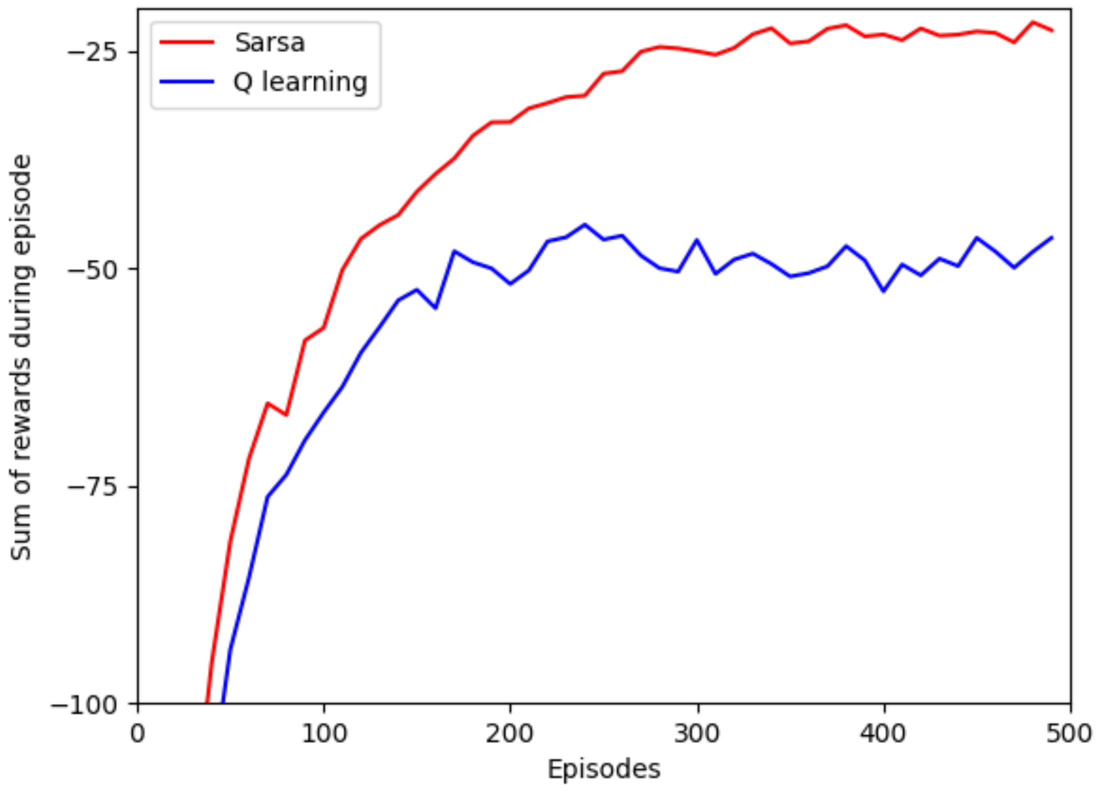
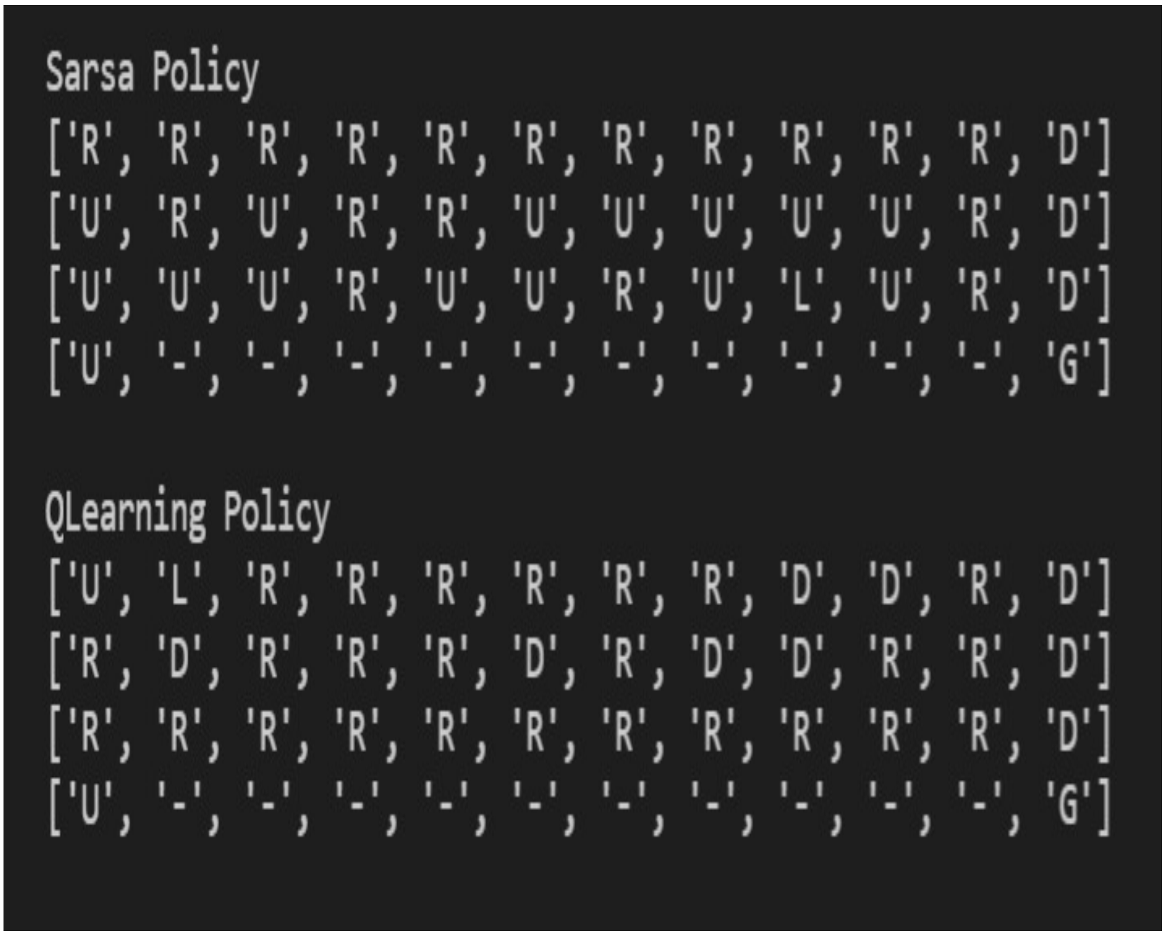


Figure 3. Epsilon Value 0.1 and Alpha 0.1

On investigating the effects of choosing different values for the exploration parameter epsilon for both methods, we found many interesting things.

- For e=0.1 as shown in the policy figure below the agent following SARSA took the safe but longer path while the agent following the Q-Learning took the shorter but dangerous path i.e., going along the cliff. The only reason for Q-Learning being penalised more is that he is falling off the grid some of the times while exploring and thus reducing its total sum of rewards.



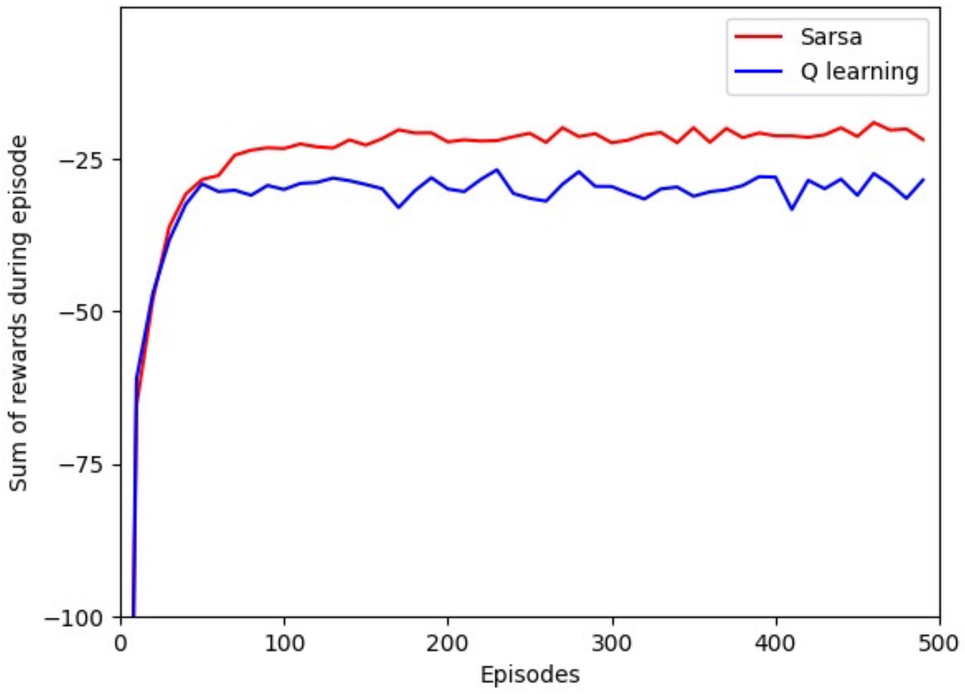
```
Sarsa Policy
['R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'D']
['U', 'R', 'U', 'R', 'R', 'U', 'U', 'U', 'U', 'U', 'R', 'D']
['U', 'U', 'U', 'R', 'U', 'U', 'R', 'U', 'L', 'U', 'R', 'D']
['U', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', 'G']

QLearning Policy
['U', 'L', 'R', 'R', 'R', 'R', 'R', 'R', 'D', 'D', 'R', 'D']
['R', 'D', 'R', 'R', 'R', 'D', 'R', 'D', 'D', 'R', 'R', 'D']
['R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'D']
['U', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', 'G']
```

Figure 4. ***Epsilon Value 0.1, Alpha 0.5***

Where U = UP, R = Right, L = Left, D = Down and G = Goal

Below are some graphs which we created to study the effects of epsilon on Q-Learning and SARSA learning techniques:-

- For both methods, on average, agent have figured out the relatively good optimal path after almost 30-40 episodes.This can be seen in the figure 4 and figure 5 when we chose extreme values of epsilon.



○

Figure 5. Epsilon Value 0.05 and Alpha 0.5

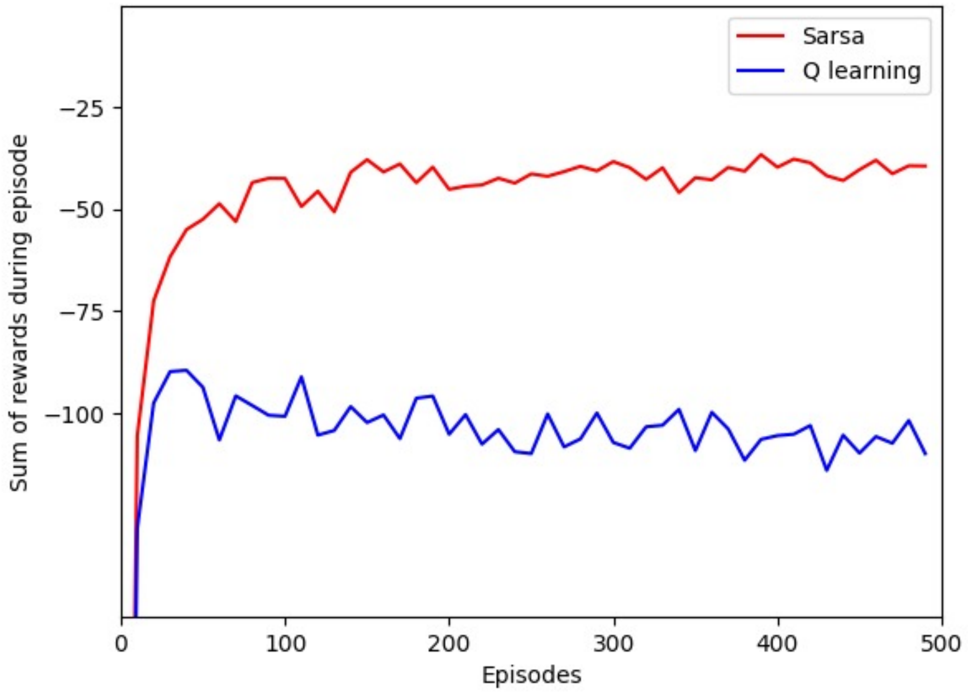


Figure 6. Epsilon Value 0.2 and Alpha 0.5

- Q Learning performed much better for low exploration values, the more we explore the more it is falling off the grid and lowering its reward.

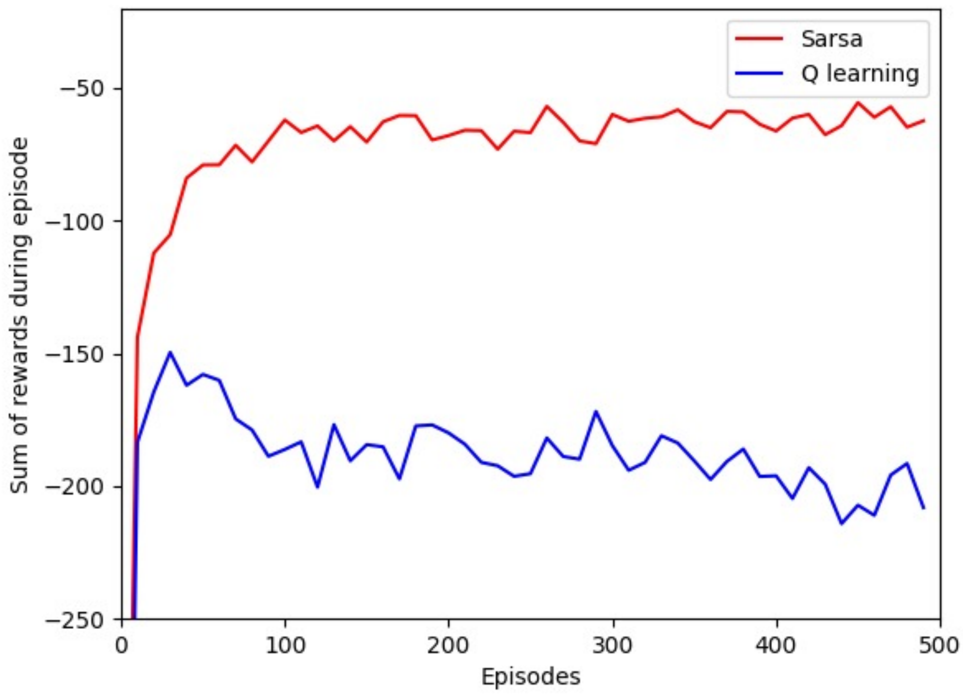


Figure 7. Epsilon Value 0.3 and Alpha 0.5

For Q learning, when the agent is exploring too often, therefore falling off the cliff while following the optimal policy route. This is due to Q learning having a more optimistic value estimation (by updating the Q values with the max value of the next state). By having lower epsilon values, the sum of rewards of Q learning improves due to it exploring less (as seen in the figure below).

For SARSA with increased exploration the agent is collecting more negative reward, but this effect is not that much prominent as compared to the Q-learning because of the policy it is following.
It is observed that a high epsilon value (exploring too much) is not as detrimental to the sum of rewards due to SARSA following a safer path, therefore when it explores, it does not fall off the cliff.

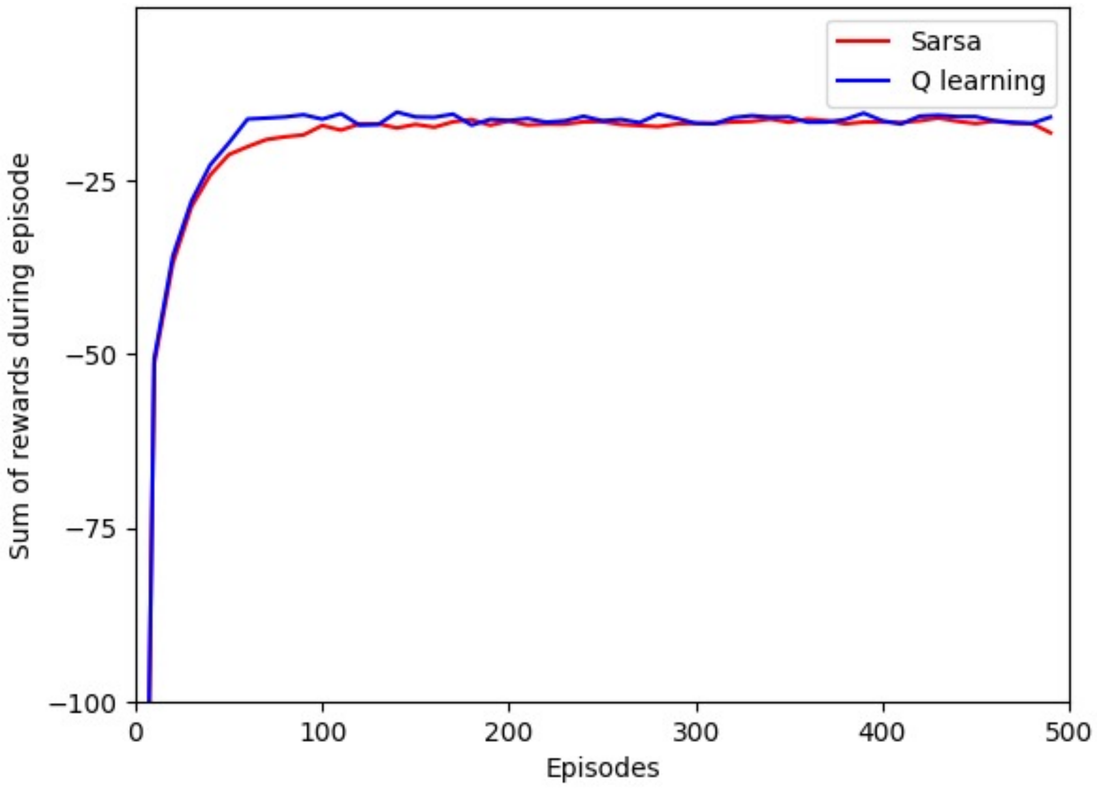


Figure 8. Epsilon Value 0.01 and Alpha 0.5

For epsilon value 0.01, the smallest, it is observed that the sum of rewards of Q learning and SARSA are similar. Furthermore, compared to the higher epsilon values, the resultant sum of rewards is the highest in the case of Q learning. As mentioned, this is due to having less exploration, so the agent falls off the cliff less.

REFERENCES

[1]M. Kusy and R. Zajdel, ‘Stateless Q-Learning Algorithm for Training of Radial Basis Function Based Neural Networks in Medical Data Classification’, in Intelligent Systems in Technical and Medical Diagnostics, Berlin, Heidelberg, 2014, pp. 267–278. doi: 10.1007/978-3-642-39881-0_22.

[2] M. Kusy and R. Zajdel, ‘Probabilistic neural network training procedure based on Q(0)-learning algorithm in medical data classification’, Appl Intell, vol. 41, no. 3, pp. 837–854, Oct. 2014, doi: 10.1007/s10489-014-0562-9.

[3]A. Nowe, P. Vrancx, and Y.-M. De Hauwere, ‘Game Theory and Multi-agent Reinforcement Learning’, in Adaptation, Learning, and Optimization, 2012, p. 30. doi: 10.1007/978-3-642-27645-3_14.

[4] S. Kapoor, ‘Multi-Agent Reinforcement Learning: A Report on Challenges and Approaches’, arXiv:1807.09427 [cs, stat], Jul. 2018, Accessed: Jun. 22, 2021. [Online]. Available: http://arxiv.org/abs/1807.09427