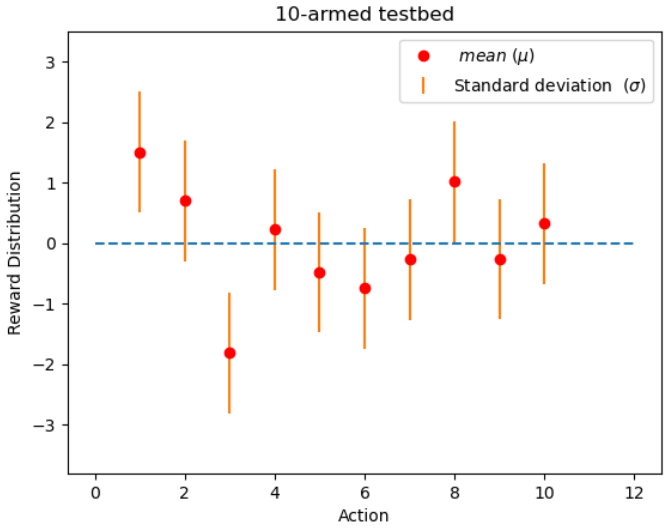
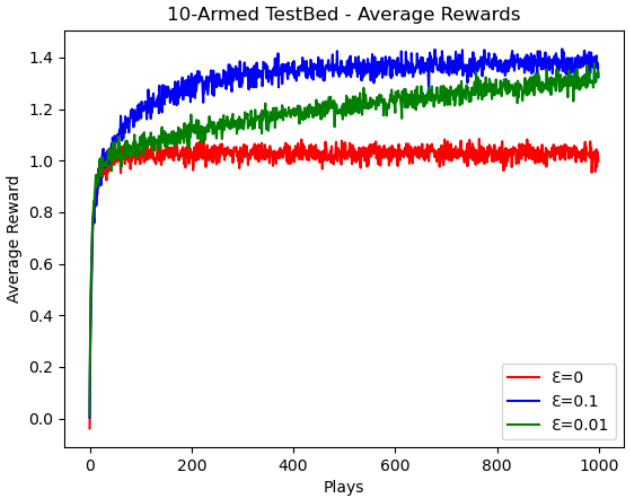
Assignment 1 – Reinforcement Learning

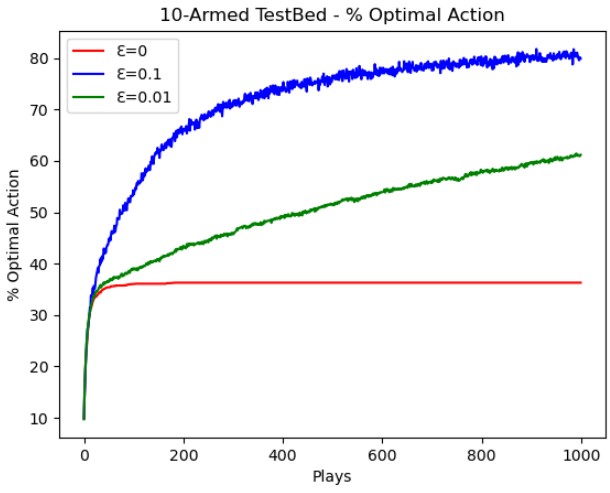
Group 7		
Srirag Jayakumar	201518670	s.jayakumar1@liverpool.ac.uk
Jishnu Prakash Kunnanath Poduvattil	201581347	j.kunnanath-poduvattil@liverpool.ac.uk
Akhil Raj	201594703	a.raj@liverpool.ac.uk

1) Problem 1

Comparison of Ɛ-Greedy Methods for values 0.01 and 0.1 is implemented in python and the source file is attached. The results are discussed below.







We recreated the same conditions required for the experiment and re-implemented the figure 2.2 in the Sutton & Barto book. Comments and documentation regarding running the code and recreating our results is explained in the source file “10armtestbed.py” and “Readme.md”.

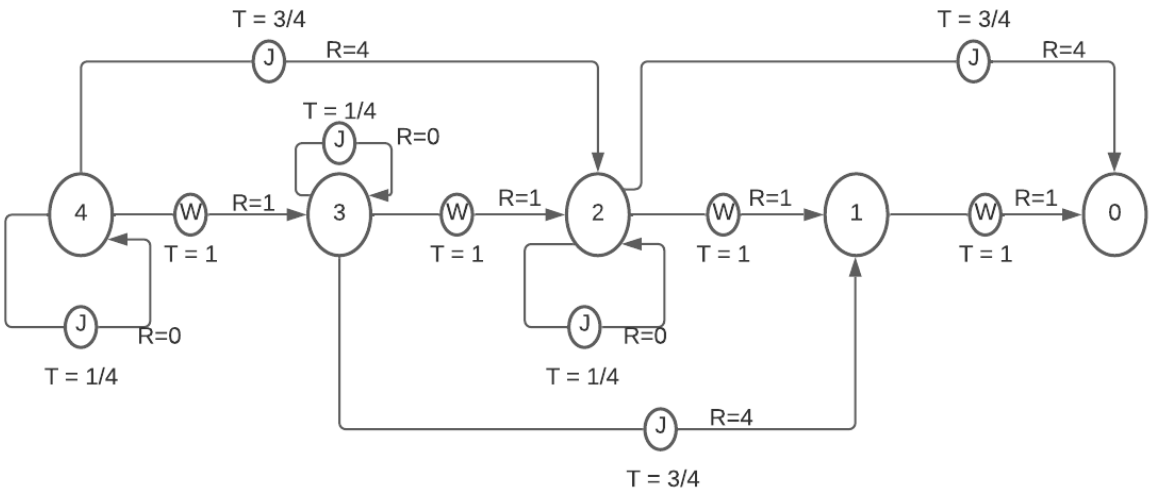
The Exploration and Exploitation Dilemma: -

The notion to take an optimal decision based on available knowledge assuming it is sufficient and the notion to go with a suboptimal decision thinking the available knowledge is insufficient, had been studied and discussed by the scientific community for a long time now and remains as first interest. This dilemma to choose between exploitation and exploration consists of a phenomenal trade-off. When a greedy strategy chooses the best action from current knowledge, the disadvantage is that the performance depends on the accuracy of the knowledge. if our knowledge is not accurate enough, the agent will be stuck choosing the suboptimal actions. In an Ɛ-Greedy strategy, the agent explores for the given probability value of Ɛ. This leads to better learning and performance in some cases.

Given the 10-armed test bed case, sample average technique is used to compute action value estimates in all 3 methods. In terms of exploitation, the greedy method (Ɛ=0) showed a fast learning at the beginning but levelled off at a later stage whereas speaking in terms of exploration, the Ɛ-greedy method shows a better performance. 1000 plays of 2000 independent runs were conducted using two different values of epsilon. For the case (Ɛ=0.1), they improved their chance of finding the optimal action by exploring. The learning was slow at first for the case (Ɛ=0.01), but eventually it performed well than the other two methods.

We can choose between exploration and exploitation with respect to the solution we want to achieve for a particular task. If the reward variances are 0, the greedy method can perform well and the noisier the rewards can get, including exploration using the Ɛ-greedy methods can do better performance.

2) Problem 2



The above diagram is drawn according to the specifications given. Rewards are calculated as per the given equation and labelled in the above diagram. As the policy is not specified, assuming that action probability is 1.

Given that, $R(s,a,s') = (s-s')^2$ for all (s,a,s')

Reward	Value
R (4, W, 3)	1
R (3, W, 2)	1
R (2, W, 1)	1
R (1, W, 0)	1
R (4, J, 2)	4
R (3, J, 1)	4
R (2, J, 0)	4
R (4, J, 4)	0
R (3, J, 3)	0
R (2, J, 2)	0

Objective is to compute $V^*(2)$ and $Q^*(3, J)$,

State Value: -

We know that,

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P(s'|s,a)\ [R(s,a,s') + \gamma V^{\pi}(s')]$$

$$V^*(s) = max_{\pi}[V^{\pi}(s)], \quad for\ every\ s\ \epsilon\ s$$

Computing value states,

$$V^*(0) = 0\ (Terminal\ state)$$

$$V^*(1) = [1 * \left[1 + \frac{1}{2} * V^*(0)\right] = 1$$

$$V^*(2) = max_{\pi}\{V_W^{\pi}(2), V_J^{\pi}(2)\}$$

$$V_W^{\pi}(2) = [1 * \left[1 + \frac{1}{2} * V^*(1)\right] = \frac{3}{2}$$

$$V_J^{\pi}(2) = \{\frac{3}{4} * \left[4 + \frac{1}{2} * V^{\pi}(0)\right] + \frac{1}{4} * \left[0 + \frac{1}{2} * V^{\pi}(2)\right]$$

$$V_J^{\pi}(2) = \{3 + \tfrac{1}{8} * V^{\pi}(2)\}$$

Solving the above,

$$V^{\pi}(2) = \frac{24}{7}$$

$$V^*(2) = max_{\pi}\left[\frac{3}{2}, \frac{24}{7}\right]$$

$$V^*(2) = \frac{24}{7}$$

Action Value: -

We know that,

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') * Q^{\pi}(s',a') \right]$$

Bellman optimality equation for action value,

$$Q^{*}(s,a) = R(s,a,s') + \gamma \sum_{s'} P(s'|s,a)\, V^{*}(s')$$

Finding $Q^{*}(3,J)$,

$$Q^{*}(3,J) = 4 + \frac{1}{2} * \left[\frac{1}{4} * V^{*}(3) + \frac{3}{4} * V^{*}(1) \right]$$

Estimating $V^{*}(3)$,

$$V^{*}(3) = V_J^{\pi}(3) = \{ \frac{3}{4} * \left[4 + \frac{1}{2} * V^{\pi}(1) \right] + \frac{1}{4} * \left[0 + \frac{1}{2} * V^{\pi}(3) \right]$$

$$V^{*}(3) = V_J^{\pi}(3) = \left\{ \frac{3}{4} * \frac{9}{2} + \frac{V^{\pi}(3)}{8} \right\}$$

$$V^{*}(3) = \frac{27}{7}$$

$$Q^{*}(3,J) = 4 + \frac{1}{2} * \left[\frac{1}{4} * \frac{27}{7} + \frac{3}{4} * 1 \right]$$

$$Q^{*}(3,J) = 4 + \frac{1}{2} * \left[\frac{27}{7} + \frac{3}{4} * 1 \right]$$

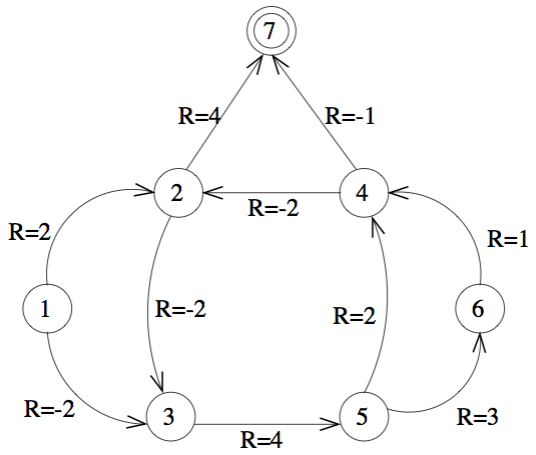
$$Q^{*}(3,J) = 4 + \frac{6}{7}$$

$$Q^{*}(3,J) = \frac{34}{7}$$

Thus computed $V^{*}(2)$ and $Q^{*}(3,J)$.

3) Problem 3

The objective is to find state values for the given reinforcement learning problem.



We know that for a given policy the state-value function estimate using temporal difference is,

$$V(S_t) = V(S_t) + \alpha[r_{t+1} + \gamma * V(S_{t+1}) - V(S_t)]$$

Given that initial value of all the states is 0, $\alpha = 0.5, \gamma = 1$.

V(1)	V(2)	V(3)	V(4)	V(5)	V(6)	V(7)
0	0	0	0	0	0	0

Episode 1: -

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

Value state	Calculation	Value
V(1) = V(1) + 0.5(R_{t+1}+1*V(3)-V(1))	0+0.5(-2+0-0)	-1
V(3) = V(3) + 0.5(R_{t+1}+1*V(5)-V(3))	0+0.5(4+0-0)	2
V(5) = V(5) + 0.5(R_{t+1}+1*V(4)-V(5))	0+0.5(2+0-0)	1
V(4) = V(4) + 0.5(R_{t+1}+1*V(2)-V(4))	0+0.5(-2+0-0)	-1
V(2) = V(2) + 0.5(R_{t+1}+1*V(7)-V(2))	0+0.5(4+0-0)	2

So after episode 1, the value states are

V(1)	V(2)	V(3)	V(4)	V(5)	V(6)	V(7)
-1	2	2	-1	1	0	0

Episode 2: -

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

Value state	Calculation	Value
V(2) = V(2) + 0.5(R_{t+1}+1*V(3)-V(2))	2+0.5(-2+2-2)	1
V(3) = V(3) + 0.5(R_{t+1}+1*V(5)-V(3))	2+0.5(4+1-2)	3.5
V(5) = V(5) + 0.5(R_{t+1}+1*V(6)-V(5))	1+0.5(3+0-1)	2
V(6) = V(6) + 0.5(R_{t+1}+1*V(4)-V(6))	0+0.5(1-1-0)	0

$V(4) = V(4) + 0.5(R_{t+1}+1*V(7)-V(4))$	-1+0.5(-1+0-1)	-2

So after episode 2, the value states are

V(1)	V(2)	V(3)	V(4)	V(5)	V(6)	V(7)
-1	1	3.5	-2	2	0	0

Episode 3: -

{5, 4, 2, 7}

Value state	Calculation	Value
$V(5) = V(5) + 0.5(R_{t+1}+1*V(4)-V(5))$	2+0.5(2-2-2)	1
$V(4) = V(4) + 0.5(R_{t+1}+1*V(2)-V(4))$	-2+0.5(-2+1+2)	-1.5
$V(2) = V(2) + 0.5(R_{t+1}+1*V(7)-V(2))$	1+0.5(4+0-1)	2.5

So after episode 3, the value states are

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

4) Problem 4

a) Q-learning update rule for stateless or 1-state problem?

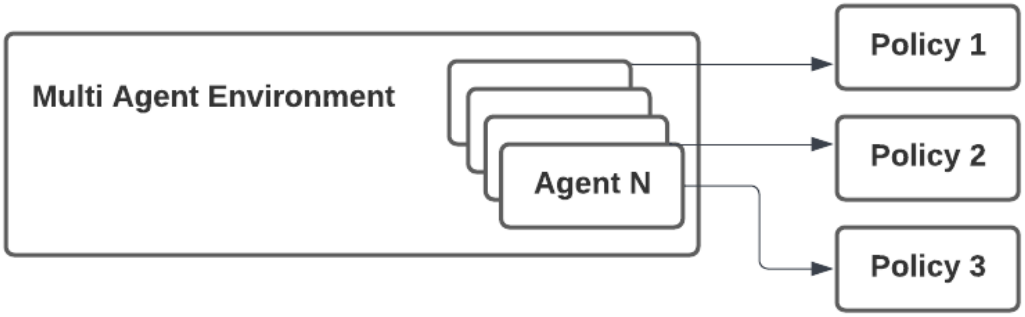
- Initially, the stateless Q-learning algorithm sets the estimates of its actions to 0
- At each iteration, it applies an action by following the ε-greedy strategy, i.e., it selects the best-rewarding action with probability $1 - \varepsilon_t$, and a random one (uniformly distributed) the rest of the times.
- After choosing action, it observes the generated reward, and updates the estimated value
- Finally, εt is updated to follow a decreasing sequence

$$\varepsilon_t = \frac{\varepsilon 0}{\sqrt{t}}$$

b) When moving from single- to multi-agent learning, one of the main challenges that arises is non-stationarity, since multiple agents are learning at the same time, the environment can be modified by the actions of all agents. From the perspective of any given agent, the world is non-stationary. To succeed, the agents need to consider the behaviour of other agents and adapt to the joint behaviour accordingly. This leads to varying learning speeds.

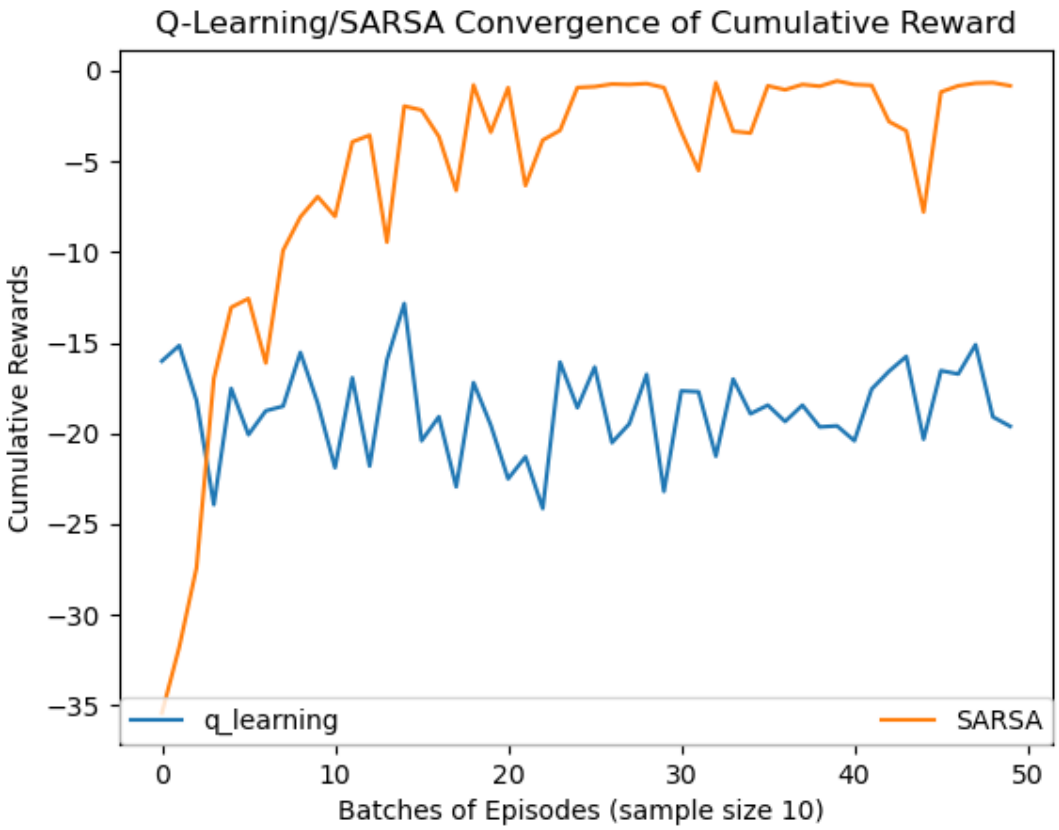
As the number of agents increases, multi-agent learning reduces the time but, it requires large amounts of computational resources and memory.

Since the actions of all agents influence the return, it is difficult for an individual agent to isolate its individual contribution to the team's success. For instance, an agent may have chosen the action that maximises reward in a given state, but the resulting return could be negative if all the other agents took exploratory actions. Thus, the agent may falsely adjust its policy to reduce the probability of selecting that action.



5) Problem 5

We recreated the same conditions for the comparison of SARSA and Q-Learning in the cliff walking task and re-implemented the results in figure 6.4 in the Sutton & Barto book. Comments and documentation regarding running the code and recreating our results is explained in the source file "tempDifference.py" and "Readme.md".



For the cliff walking task, we used temporal difference method as the prime action – state value update method where an agent explores the environment by taking action following an ϵ-greedy policy and based on its latest observation, which is summarised as a value of state-action, it updates its current estimates by tweaking the current estimation a little bit towards the latest observation (1-step TD). SARSA and Q – learning are the two types of temporal difference methods and the difference lies in how the Q function is updated.

SARSA is an on-policy technique where we still follow the ϵ-greedy policy to get to the next state and to update the Q value of the current state which makes SARSA a conservative agent. As seen in the cliff walking task, SARSA always prefers the longer safer route so as to avoid falling down the cliff. As a result, SARSA keeps on exploring and finds the near optimal solution thus taking a longer time to attain the results (observed in the graph above). Thus SARSA would be a better option in a situation where mistakes are costly and time is not of the essence.

Q- learning on the other hand is an off- policy technique where it updates the Q-function based on the maximum Q-value of the next state thus making it a more aggressive agent. In the cliff walking task, Q-learning preferred the shortest, optimal path thus attaining the solution and rewards faster as compared to SARSA. Thus Q-learning would be the best option in a fast – iterating environment where time is of the essence and mistakes can be somewhat tolerated.

It can be observed from the code that by reducing the ϵ value from 0.1, the greediness of both the algorithm especially SARSA increases and gradually both methods would asymptotically converge to the optimal policy.