# CS6700 : Reinforcement Learning
Written Assignment #1

**Topics**: Intro, Bandits, MDP, Q-learning, SARSA, FA, DQN **Deadline**: 22/03/2024, 23:59

**Name:** Janmenjaya Panda **Roll number:** ME20B087

- This is an individual assignment. Collaborations and discussions are strictly prohibited.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- Type your solutions in the provided LaTeXtemplate file.
- **Please start early.**

1. (3 marks) [TD, IS] Consider the following deterministic grid-world.

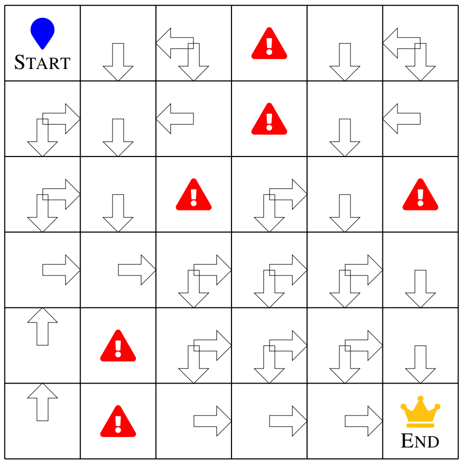


Figure 1: A deterministic grid-world with an optimal policy

Every actions yields a reward of -1 and landing in the red-danger states yields an additional -5 reward. The optimal policy is represented by the arrows. Now, can you learn a value function of an arbitrary policy while strictly following the optimal policy? Support your claim.

**Solution:**

**Claim 0.1.** *For $\pi$ being the target policy and for $\mu$ being the behaviour policy, if there exists some action $a$ for some state $s$ such that, $\pi(a|s) > 0$, but $\mu(a|s) = 0$, in other words, there exists an action taken under the target policy that is never taken under the behaviour policy, then the agent will never learn the target policy.*

**Proof.** In off-policy TD learning, we have two key policies:

*Target Policy:* This is the policy that we want to learn and improve. The target policy is the one that we ultimately want the agent to follow to maximize rewards in the environment.

*Behavior Policy:* This is the policy that the agent follows to interact with the environment and gather experience. The behaviour policy can be anything - it could be a random policy, a deterministic policy, an optimal policy, or even another policy being learned.

Consider the update rule of action-value functions in TD(0) under target policy $\pi$:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)]$$

In TD learning, the agent updates its estimate of the value of a state-action pair based on the observed reward and the estimate of the value of the next state-action pair.

If the behaviour policy $\mu$ never selects certain actions that the target policy $\pi$ would select, the agent will never experience those state-action pairs. As a result, the agent's estimate of the value of those state-action pairs will never be updated, and it will remain at its initial value (usually initialized to 0 or some other arbitrary value).

Consequently, if the target policy $\pi$ differs significantly from the behaviour policy $\mu$ and the agent never explores the state-action pairs that the target policy $\pi$ would have selected, the learned Q-values (or action-value estimates) will be biased towards the behaviour policy $\mu$. In other words, the agent will effectively learn the value of actions under the behaviour policy $\mu$ rather than under the target policy $\pi$.

Also, Importance sampling, that is one of methods for off-policy learning, assumes that for all action $a$ and for some state $s$, if $\pi(a|s) > 0 \implies \mu(a|s) > 0$. We shall show that the violation of this rule, that is if there is some action for which $\pi(a|s) > 0$ and $\mu(a|s) = 0$, it will lead to some absurdity. The reader may recall that the importance sampling ratio is given as follows:

$$\rho_t^T = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{\mu(a_k|s_k)}$$

and the weighted average return for $V$ under $\pi$ wrt $\mu$ is given as follows:

$$V(s) = \frac{\sum\limits_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum\limits_{t \in \mathcal{T}(s)} \rho_t^{T(t)}}$$

where $\mathcal{T}(s)$ are the set of times in which state $s$ gets visited.

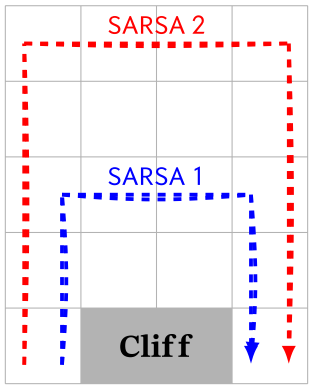
For the given condition of existence of an action $a$ for which $\pi(a|s) > 0 = \mu(a|s)$, **the importance sampling ratio will be undefined**. Thus, it is impossible to learn the target policy $\pi$ from the behaviour policy $\mu$. $\square$

Consider the optimal policy $\boldsymbol{\pi}_*$ as the behaviour policy that is shown in Figure 1 and an arbitrary policy $\boldsymbol{\pi}$ as the target policy. In conclusion, there are two possible scenarios based on the probability of an action $a$ under $\pi$ and under $\pi_*$.

1. If there exists an action $a$ such that for some state $s$, it holds that $\pi(a|s) > 0 = \pi_*(a|s)$, then by Claim 0.1, we conclude that $V_\pi(s)$ and $Q_\pi(s, a)$ can not be learnt.

2. On the other hand, if for each action $a$ and for each state $s$, if $\pi(a|s) > 0$ implies $\pi_*(a|s) > 0$, then such a policy $V_\pi(s)$ and $Q_\pi(s, a)$ can be learnt using importance sampling. Note that, the set of paths traced by policy $\pi$ is a subset of the set of paths traced by the optimal policy $\pi_*$ shown in Figure 1. Also, from the symmetry of states, actions and rewards, in this setting, the reader may easily verify that $\pi$ will eventually become another optimal policy.

2. (1 mark) [SARSA] In a 5 x 3 cliff-world two versions of SARSA are trained until convergence. The sole distinction between them lies in the $\epsilon$ value utilized in their $\epsilon$-greedy policies. Analyze the acquired optimal paths for each variant and provide a comparison of their $\epsilon$ values, providing a justification for your findings.



**Solution:**
Recall the $\epsilon$-greedy policy:
Given $Q$ and $\epsilon$, the $\epsilon$-greedy policy balances exploration and exploitation by choosing the action with the highest $Q$−value with probability $1 - \epsilon$ (aka exploitation) and selecting a random action with probability $\epsilon$ (aka exploration); in other words, it selects:

$$A = \begin{cases} \text{An arbitrary action,} & \text{with probability } \epsilon \\ \text{The action with maximum } Q, & \text{with probability } 1 - \epsilon \end{cases}$$

where $\epsilon$ is the exploration rate.

In the given scenario, we have two versions of SARSA trained until convergence in a $5 \times 3$ cliff-world environment. The only difference between these two versions is the $\epsilon$ value used in their $\epsilon$-greedy policies. SARSA-1 shows a shorter optimal path, whereas SARSA-2 acquires a relatively longer optimal path. Let's analyze the acquired optimal paths for each variant and compare their $\epsilon$ values:

1. *Shorter Optimal Path Variant (SARSA 1):*
   - This variant used a lower $\epsilon$ value in its $\epsilon$-greedy policy.
   - A lower $\epsilon$ value means that the agent is more likely to exploit known information rather than explore.
   - The agent in this variant is less inclined to take random actions and is more likely to follow the learned optimal policy.
   - Consequently, the agent is more likely to select actions that lead to the shorter optimal path rather than exploring alternative, longer paths or suboptimal paths.
   - Therefore, the shorter path variant is likely to result from the agent's tendency to exploit known information and follow the learned optimal policy due to the lower $\epsilon$ value.
2. *Longer Optimal Path Variant (SARSA 2):*
   - This variant used a higher $\epsilon$ value in its $\epsilon$-greedy policy.
   - A higher $\epsilon$ value means that the agent is more likely to explore rather than exploit.
   - Consequently, the agent in this variant is more inclined to take random actions, even if they are longer or suboptimal.
   - This increased exploration can lead the agent to discover longer, sometimes suboptimal paths while searching for the optimal path.
   - Therefore, the longer optimal path variant is likely to result from the agent's increased exploration due to the higher $\epsilon$ value.

In summary, the comparison of the acquired optimal paths for each variant of SARSA indicates that the $\epsilon$ value used in the $\epsilon$-greedy policy significantly influences the exploration -exploitation trade-off. A higher $\epsilon$ value leads to increased exploration, which may result in discovering longer, sometimes suboptimal paths, while a lower $\epsilon$ value favours the exploitation of known information and may tend to result in shorter, optimal paths. Thus

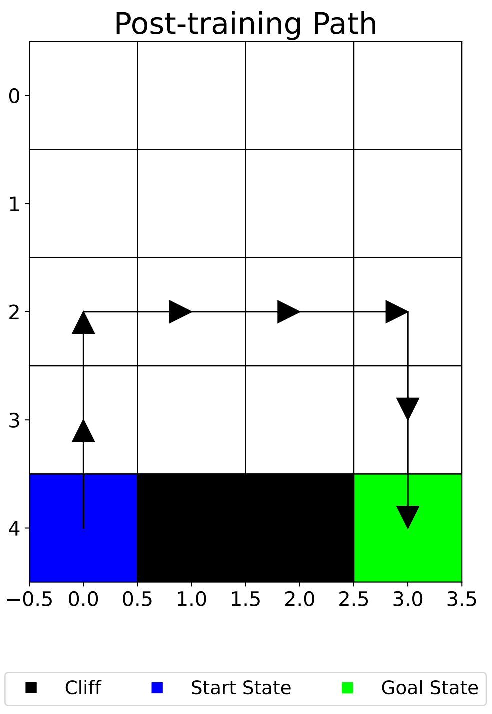
$$\boxed{0 \leqslant \epsilon_{SARSA_1} < \epsilon_{SARSA_2} \leqslant 1}$$

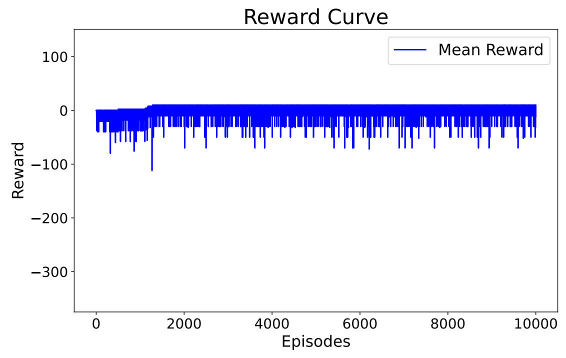
**Appendix:**
The experiment is recreated with the bottom-left corner as the start state and the bottom-right corner as the goal state. Any transition that lands the agent in the Cliff earns a reward of -100, whereas any transition that lands the agent in the goal state earns a reward of +10. All other in-world transition earns 0 rewards. We run two different simulations of SARSA with $\epsilon$-greedy policy for 5 runs each of 10000 episodes for two values of $\epsilon$, that is $\epsilon = 0.15$ and $\epsilon = 0.5$, and the following plots capture an instance of the optimal path taken by the agent after the training period. It has been inspired by Programming Assignment 1 of CS6700 Reinforcement Learning Spring 2024 Offering.

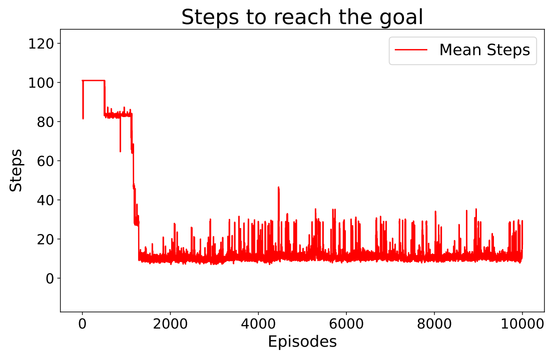
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(4,0)$ |
| Algorithm | SARSA |
| **Policy** | **$\epsilon-$greedy** |

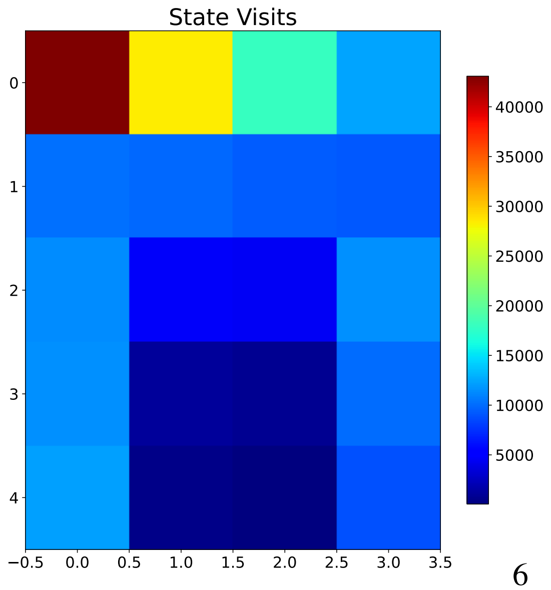
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.95 |
| **Epsilon** ($\epsilon$) | **0.15** |

| | |
|---|---|
| Reward | 10 |
| Path taken | [40, 30, 20, 21, 22, 23, 33, 43] |









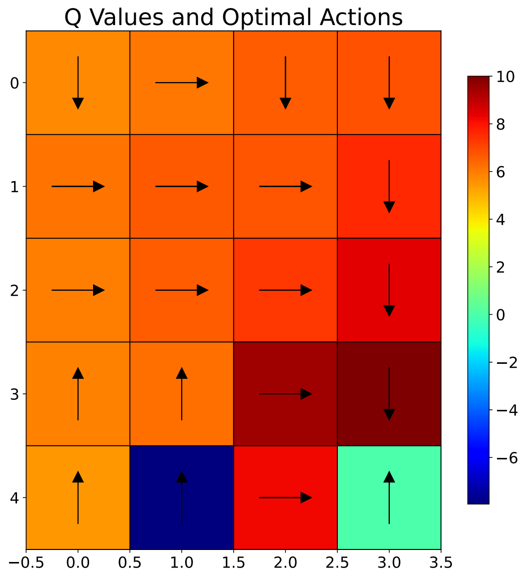
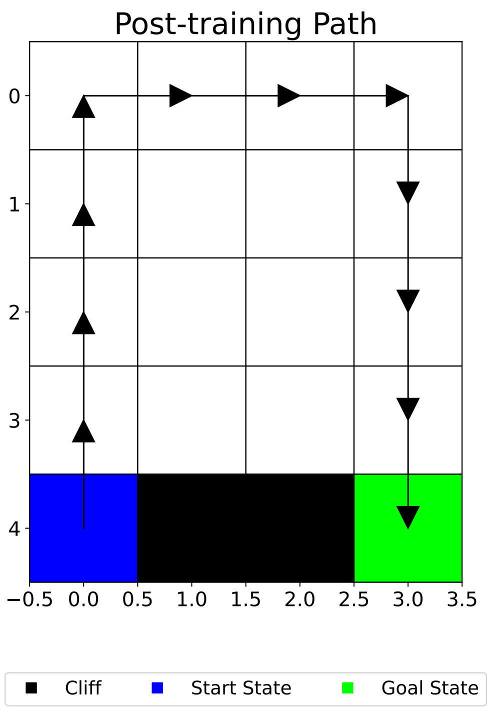


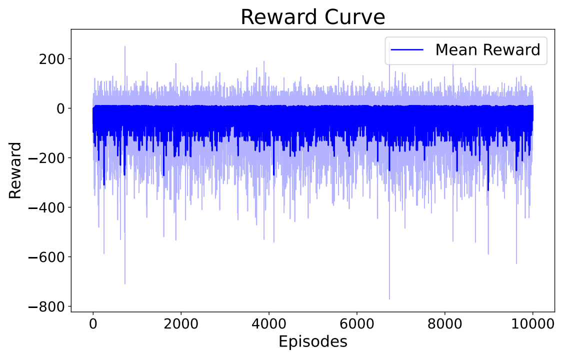
Table 1: $\boxed{\epsilon : \mathbf{0.15}}$

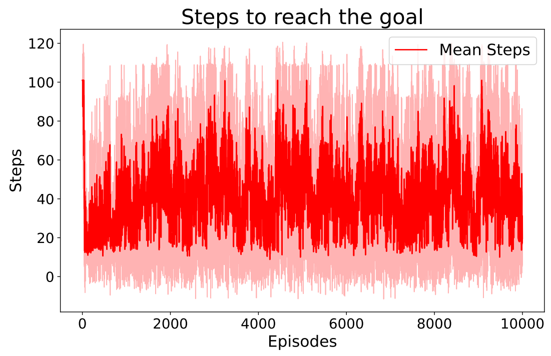
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(4, 0)$ |
| Algorithm | SARSA |
| **Policy** | **$\epsilon$−greedy** |

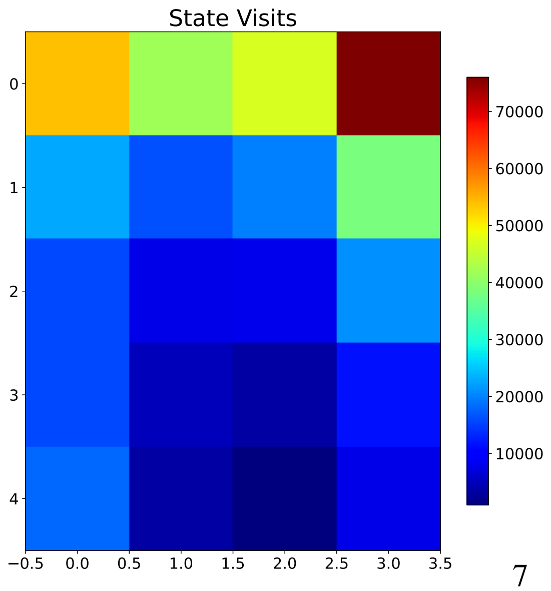
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.95 |
| **Epsilon** ($\epsilon$) | **0.5** |

| | |
|---|---|
| Reward | 10 |
| Path taken | [40, 30, 20, 10, 00, 01, 02, 03, 13, 23, 33, 43] |









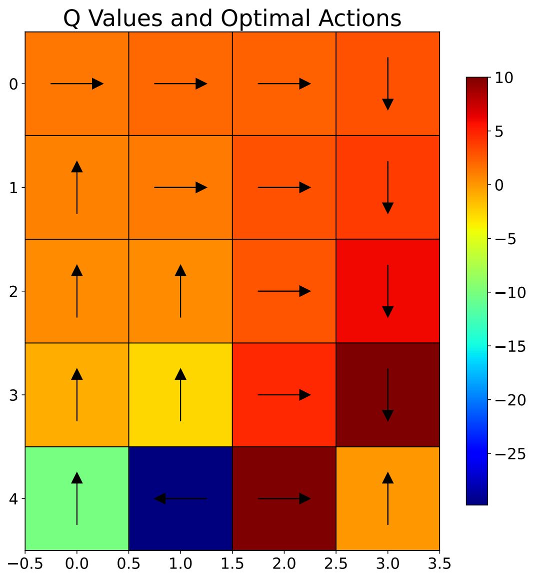
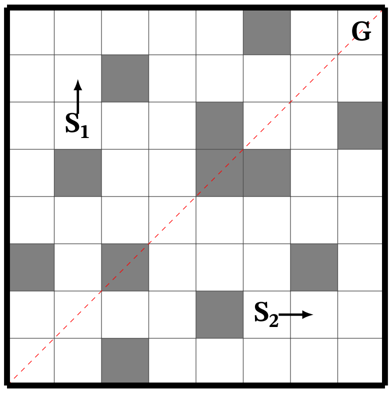


Table 2: $\boxed{\epsilon : \mathbf{0.5}}$

3. (2 marks) [SARSA] The following grid-world is symmetric along the dotted diagonal. Now, there exists a symmetry function $F : S \times A \rightarrow S \times A$, which maps a state-action pair to its symmetric equivalent. For instance, the states $S_1$ and $S_2$ are symmetrical and F($S_1$,North) = $S_2$,East.



Given the standard SARSA pseudo-code below, how can the pseudo-code be adapted to incorporate the symmetry function $F$ for efficient learning?

**Algorithm 1 :** SARSA Algorithm

```
1:  Initialize $Q$-values for all state-action pairs arbitrarily
2:  for each episode do
3:      Initialize state $s$
4:      Choose action $a$ using $\epsilon$-greedy policy based on $Q$-values
5:      while not terminal state do
6:          Take action $a$, observe reward $r$ and new state $s'$
7:          Choose action $a'$ using $\epsilon$-greedy policy based on $Q$-values for state $s'$
8:          $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
9:          $s \leftarrow s'$, $a \leftarrow a'$
10:     end while
11: end for
```

**Solution:** To improve the SARSA algorithm's learning efficiency within a symmetric grid environment, we introduced a symmetry function (referred to as $F$) into the update process. This enhancement enables us to capitalize on the symmetrical equivalence among state - action pairs, reducing redundancy in learning efforts and effectively streamlining the exploration of the state - action space.

In the Symmetric SARSA Algorithm 2, following the standard Q-value update for the original state-action pair $(s, a)$, we employ the symmetry function $F$ to **identify its unique symmetrical counterpart** $(\mathbf{s_F}, \mathbf{a_F})$. By **updating the Q-value for both the original and**

**the symmetric pairs**, we ensure that insights gained in one section of the grid are efficiently transferred to its symmetrical counterpart. Note that since the environment is symmetric, we assume that **symmetric actions from symmetric states will generate precisely the same reward**.

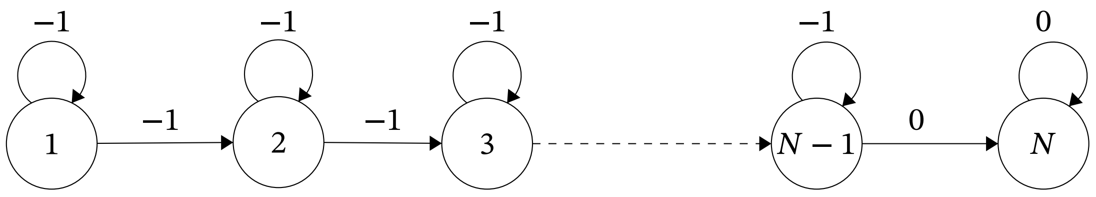
Moreover, to appropriately merge information from both the original and symmetric pairs, we **compute the average of their updated Q-values**. This averaging mechanism guarantees that the learning process properly acknowledges the symmetry between state-action pairs, facilitating the convergence of Q-values.

The newly introduced lines in Algorithm 2 as compared to Algorithm 1 are shown blue.

**Algorithm 2 :** Symmetric SARSA Algorithm

```
 1: Initialize $Q$-values for all state-action pairs arbitrarily
 2: for each episode do
 3:     Initialize state $s$
 4:     Choose action $a$ using $\epsilon$-greedy policy based on $Q$-values
 5:     while not terminal state do
 6:         Take action $a$, observe reward $r$ and new state $s'$
 7:         Choose action $a'$ using $\epsilon$-greedy policy based on $Q$-values for state $s'$
 8:         $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma Q(s',a') - Q(s,a))$
 9:         $(s_F, a_F) \leftarrow F(s,a)$
10:         $Q(s_F,a_F) \leftarrow Q(s_F,a_F) + \alpha\left(r + \gamma Q(s'_F,a'_F) - Q(s_F,a_F)\right)$
11:         $Q(s,a) \leftarrow \dfrac{Q(s,a) + Q(s_F,a_F)}{2}$
12:         $Q(s_F,a_F) \leftarrow Q(s,a)$
13:         $s \leftarrow s'$, $a \leftarrow a'$
14:     end while
15: end for
```

4. (4 marks) [VI] Consider the below deterministic MDP with $N$ states. At each state there are two possible actions, each of which deterministically either takes you to the next state or leaves you in the same state. The initial state is 1 and consider a shortest path setup to state $N$ (Reward -1 for all transitions except when terminal state $N$ is reached).



Now on applying the following Value Iteration algorithm on this MDP, answer the below questions:

**Algorithm 3 :** Value Iteration Algorithm

```
1: Initialize $V$-values for all states arbitrarily over the state space $S$ of $N$ states. Define
   a permutation function $\phi$ over the state space
2: $i \leftarrow 0$
3: while NotOptimal(V) do
4:     $s \leftarrow \phi((i \mod N) + 1)$
5:     $V(s) \leftarrow \max_a \sum_{s,r'} p(s', r|s, a)[r + \gamma * V(s')]$
6:     $i \leftarrow i + 1$
7: end while
```

**Solution:**
**Notation 4:**

1. $[N] := \{1, 2, 3, \ldots, N\}$
2. $V_k(s)$ :=Value function of state $s$ at iteration $k$, for some $s \in [N]$ and for some non-negative integer $k$.
3. $V_*(s)$ := the optimal value function of state $s$ for some $s \in [N]$.

**Assumption 4:**

1. Transition to the terminal state $N$ earns zero reward.

   **Claim 0.2.** *Either $\gamma = 1$ or $V_*(N) = 0$ or possibly both.*

   **Proof.** Note that any policy shall trace the self-loop action at state $N$ as that is the only available action at that state. By Bellman Optimality equation for state $N$,

   $$V_*(N) = 0 + \gamma V_*(N).$$

   Thus, either $\gamma$ is one or $V_*(N)$ is zero or possibly both. □

2. There is some small positive constant $\epsilon$ such that for each state $s$, a value function $V(s)$ is considered optimal if $|V(s) - V_*(s)| \leqslant \epsilon$ and we run Algorithm 3 till the subsequent difference between $V(s)$, for all state $s \in [N]$, is less than this threshold $\epsilon$.
3. Clearly, if $\gamma = 1$, then $|V_o(N) - V_*(N)| \leqslant \epsilon$.
4. $\gamma$ is not one for a deterministic policy in which the agent takes at least one self-loop in some non-terminal state.

**Proof.** Suppose not; let $s$ be a non-terminal state in which the agent takes the self-loop under some deterministic policy. Then by Bellman Optimality equation:

$$V_*(s) = -1 + V_*(s)$$

which is, clearly, not true for any real value $V_*(s)$. □

(a) (1 mark) Design a permutation function $\phi$ (which is a one-one mapping from the state space to itself, defined here), such that Algorithm 3 converges the fastest and reason about how many steps (value of $i$) it would take.

**Solution:** We adopt Notation 4 and Assumption 4.
Consider the permutation function:$\phi_1 : [N] \to [N]$ defined as follows:

$$\phi_1(i) = N + 1 - i \quad \forall\, i \in [N] \tag{1}$$

**Claim 0.3.** *For some given arbitrary initialization of value functions, the permutation function given in Equation 1 ensures the fastest convergence of the value function through Algorithm 3.*

**Proof.** Consider the terminal state $N$. Clearly, under any policy, the agent always follows a single action at state $N$, that is – self-looping back to state $N$ itself, as that is only possible action available at state $N$. For $\gamma = 1$, as per Assumption 4.3, the convergence from $V_o(N)$ to $V_*(N)$ happens in a single step, Thus, we let $\gamma < 1$. At $k$th iteration $V_k(N) = \gamma^k V_o(N)$. Thus, for convergence

$$\begin{aligned}
\gamma^k |V_o(N)| &\leqslant \epsilon \\
\implies \gamma^k &\leqslant \frac{\epsilon}{|V_o(N)|} \\
\implies k \log\gamma &\leqslant \log\frac{\epsilon}{|V_o(N)|} \\
\implies k &\geqslant \frac{\log\left(\frac{\epsilon}{|V_o(N)|}\right)}{\log\gamma}
\end{aligned}$$

Thus, if we let $C_N := \left\lceil \frac{\log\left(\frac{\epsilon}{|V_o(N)|}\right)}{\log\gamma} \right\rceil$, the terminal state value function reaches the optimal value in at least $C_N$ steps. Now, let's talk about the non-terminal state $s$. Note that the policy considered is deterministic. So, at some state $s$, for some $s < N$, under any deterministic policy the agent can execute one of the following actions:

(a) Go to state $s+1$ with probability 1, or otherwise

(b) Go back to state $s$ with probability 1.

We have two cases based on the deterministic action that the agent takes at state $s$.

Case 1: At a non-terminal state $s$, the agent self-loops back to state $s$ with probability 1. By Assumption 4.4, we let $\gamma < 1$. The Bellman Optimality equation reads as follows:

$$V_*(s) = -1 + \gamma * V_*(s)$$

Hence, $V_*(s)$ is given by $\frac{-1}{1-\gamma}$, which is clearly negative. For convergence in $k$-steps, we want

$$\begin{aligned} &|V_k(s) - V_*(s)| \leqslant \epsilon \\ \implies &\left| V_k(s) - \frac{-1}{1-\gamma} \right| \leqslant \epsilon \\ \implies &\left| V_k(s) + \frac{1}{1-\gamma} \right| \leqslant \epsilon \end{aligned}$$

In other words,

$$\frac{-1}{1-\gamma} - \epsilon \leqslant V_k(s) \leqslant \frac{-1}{1-\gamma} + \epsilon$$

For small enough $\epsilon$, it holds that $\frac{-1}{1-\gamma} + \epsilon < 0$; hence, we deduce that for convergence of $V(s)$ in $k$ steps, $V_k(s) < 0$.
At $k$th state the value function $V_k(s)$ is given by:

$$V_k(s) = \underbrace{-1 + \gamma(-1 + \gamma(-1 + \cdots + \gamma(-1 + V_o(N))))}_{k \text{ times}}.$$

Clearly, $V_k(s) = -1 + \gamma * V_{k-1}(s) \leqslant \gamma V_{k-1}(s)$. Through induction on $k$, we conclude that $V_k(s) \leqslant \gamma^k V_o(s)$. We assume that $|V_k(s)| \leqslant \gamma^k |V_o(s)|$.
Consequently,

$$\left| V_k(s) + \frac{1}{1-\gamma} \right| \leqslant |V_k(s)| + \frac{1}{1-\gamma} \leqslant \gamma^k \left| V_o(s) \right| + \frac{1}{1-\gamma}.$$

Also,

$$\begin{aligned} &\gamma^k \left| V_o(s) \right| + \frac{1}{1-\gamma} \leqslant \epsilon \\ \implies &\left| V_k(s) + \frac{1}{1-\gamma} \right| \leqslant \epsilon \end{aligned}$$

which guarantees the convergence of $V(s)$.
Consequently,

$$\begin{aligned}
&\gamma^k |V_o(s)| + \frac{1}{1-\gamma} \leqslant \epsilon \\
&\implies \gamma^k |V_o(s)| \leqslant \epsilon - \frac{1}{1-\gamma} \\
&\implies \gamma^k \leqslant \frac{\epsilon - \dfrac{1}{1-\gamma}}{|V_o(s)|} \\
&\implies k \log \gamma \leqslant \log \frac{\epsilon - \dfrac{1}{1-\gamma}}{|V_o(s)|} \\
&\implies k \geqslant \frac{\log\left(\dfrac{\epsilon - \dfrac{1}{1-\gamma}}{|V_o(s)|}\right)}{\log \gamma}
\end{aligned}$$

Let $C_s \ := \left\lceil \dfrac{1}{\log\gamma} * \log\left(\dfrac{\dfrac{1}{1-\gamma} - \epsilon}{|V_o(s)|}\right)\right\rceil$. Thus, if we assume that for some state, self-looping back is the only possible deterministic action, for that state, any permutation function ensures the convergence of the value function $V(s)$ to the optimal value function $V_*(s)$ in at least $C_s$ no. of steps as the individual step update rule of $V(s)$ is independent of $V(s')$ for any $s' \neq s$. Thus, in this case for the convergence of a state value function $V(s)$, where self-loop is the only possible deterministic action, any permutation is optimal; consequently, the permutation function 1 is also optimal.

Case 2: At a non-terminal state $s$, the agent goes to state $s+1$ with probability 1.
By the Bellman optimality equation:

$$V_*(s) = \begin{cases} -1 + \gamma * V_*(s+1) & \text{if } s < N-1 \\ \gamma * V_*(s+1) & \text{if } s = N-1 \end{cases}$$

In other words,

$$V_*(s+1) = \begin{cases} \dfrac{V_*(s)+1}{\gamma} & \text{if } s < N-1 \\ \dfrac{V_*(s)}{\gamma} & \text{if } s = N-1 \end{cases}$$

Thus, for some positive integer $k$,

$$\begin{aligned} \left|V_*(s+1) - V(s+1)\right| &\leqslant \epsilon \\ \implies \left|\frac{V_*(s) - V(s)}{\gamma}\right| &\leqslant \epsilon \\ \implies \left|V_*(s) - V(s)\right| &\leqslant \gamma\epsilon \\ \implies \left|V_*(s) - V(s)\right| &\leqslant \epsilon \end{aligned}$$

Thus, under the deterministic policy, if the agent goes from state $s$ to state $s+1$ with probability 1, at some point if $V_k(s+1)$ gets converged implies that $V_k(s)$ also converges in the same iteration if $\phi^{-1}(s+1) < \phi^{-1}(s)$, or otherwise $V_{k+1}(s)$ converges in the next iteration taking at most $N$ more steps if $\phi^{-1}(s+1) > \phi^{-1}(s)$.

Note that the permutation function 1, $\phi_1^{-1}(i) = N + 1 - i = \phi_1(i)$ for some $i \in [N]$. Also, this permutation function ensures that in one iteration, $V(s)$ gets updated immediately after the updation of $V_(s+1)$. Thus, for each of the state $s$ with $s < N$ so that the agent reaches state $s+1$ from state $s$ under the deterministic policy, the same iteration which obtains the $V_*(s+1)$ will also obtain $V_*(s)$ within one step after updating $V(s+1)$, which is the fastest possible convergence that may happen.

Conclusively, we may conclude that for some arbitrary initialization, permutation function 1 ensures the fastest possible convergence under the assumption that the policy is deterministic regardless of whether an action takes the agent from state $s$ to itself with probability one or to state $s+1$ with probability one. □

**Claim 0.4.** *Under the permutation function 1, for some given arbitrary initialization, the no. of steps taken by Algorithm 3 $\leqslant CN$ for some constant $C$.*

**Proof.** In Claim 0.3, we saw that for terminal state $N$, then the value function $V(N)$ converges to $V_*(N)$ in at least $C_N$ steps, where $C_N = \left\lceil \frac{\log\left(\frac{\epsilon}{|V_o(N)|}\right)}{\log \gamma} \right\rceil$.

Analogously, for a non-terminal state $s$, if the agent goes back to state $s$ itself under the deterministic policy, then $V(s)$ converges to $V_*(s)$ in at least $C_s$ number of steps. The expressions for $C_s$ are given in Claim 0.3.

For a non-terminal state $s$, if it goes to state $s+1$ under the deterministic policy, then $V(s)$ converges to $V_*(s)$ in the same iteration in which $V(s+1)$ converges to $V_*(s+1)$, just after one step of the convergence of the latter due to the permutation function defined.

Putting everything together, note that there exists a $C$ that satisfies the following condition:

$$\max\left(C_N, \max_{i=1}^{N-1} C_s\right) \leqslant C.$$

This $C$ ensures the convergence of the state-value function $V(s)$ to $V_*(s)$ for each of $s \in [N]$. This is the required $C$.

For each $s$, value function $V(s)$ converges in $C$ steps. Thus, for all $N$ states the total number of steps taken by the algorithm is **upper bounded by CN**.

□

(b) (1 mark) Design a permutation function $\phi$ such that the VI algorithm would take the most number of steps to converge to the optimal solution and again reason how many steps that would be.

**Solution:** Consider the permutation function:$\phi_2 : [N] \to \langle N \rangle$ defined as follows:

$$\phi_2(i) = i \quad \forall\, i \in [N] \tag{2}$$

**Claim 0.5.** *For some given arbitrary initialization, the permutation function given in Equation 2 ensures the most number of steps for convergence of the value function through Algorithm 3.*

**Proof.** In Claim 0.3, we saw that for a state $s$, if the deterministic action is to take the self-loop and go back to step $s$, for that state it holds that $|V_k(s) - V_*(s)| \leqslant \epsilon$ for some constant $k$ which is only dependent on $\gamma, V_o(s)$ and $\epsilon$ and is independent of permutation function.

On the other hand, consider a state $s$, from which the only possible action is to go to state $s+1$. Clearly, as we have seen in Claim 0.3, the value function $V(s)$ converges to $V_*(s)$ after the convergence of $V(s+1)$ to $V_*(s+1)$. But note that, under the permutation function 2, the algorithm visits $V(s)$ after one complete iteration of $N-1$ steps after updating $V(s+1)$, which is the worst case possible.

Conclusively, we may conclude that for some arbitrary initialization, permutation function 2 ensures the convergence of value functions in the most number of steps possible. □

**Claim 0.6.** *Under the permutation function 2, for some given arbitrary initialization, the no. of steps taken by Algorithm 3 $\leqslant CN^2$ for some constant $C$.*

**Proof.** In Claim 0.4, we saw that for all state $s$, the value function $V(s)$ converges to $V_*(s)$ in at most $CN$ steps, for some constant $C$ (the expression of $C$ is given in the proof of the same).

In Claim 0.5, we saw that if the deterministic policy takes the agent from state $s$ to state $s+1$, then it takes one more complete iteration for $V(s)$ to get updated after the updation of $V(s+1)$.

So for at most $N-1$ number of in between states, the total number of steps in which each of $V(s)$ gets converged is **upper bounded by CN²**.

Please refer to the proof of Claim 0.4, to see the expression of $C$. □

(c) (2 marks) Finally, in a realistic setting, there is often no known semantic meaning associated with the numbering over the sets and a common strategy is to randomly sample a state from $S$ every timestep. Performing the above algorithm with $s$ being a randomly sampled state, what is the expected number of steps the algorithm would take to converge?

**Solution:**

**Claim 0.7.** *Under an (unknown) arbitrary permutation function and for some given arbitrary initialization, the expected no. of steps taken by Algorithm 3 is upper bounded by $CN^2$ for some constant $C$.*

**Proof.** Note that from Claim 0.3 and Claim 0.6, we observed that for all state $s$, if the deterministic action taken, is to take the self-loop and go back to the same state $s$, it can be achieved in a total time of $CN$ for some constant $c$.

The overall complexity depends on the number of steps it took to update such state $s$, where the action deterministically takes it to state $s+1$.

For some permutation function $\phi$, we define forward distance $d_\phi : S \to S$ as follows. $d_\phi(x, y) :=$ the number of steps the Algorithm 3 take to update state value function $V(y)$ after state value function $V(x)$ under the permutation function $\phi$. For example, for the permutation function 1, we update $V(s)$ just after one step of update of $V(s+1)$, so $d_{\phi_1}(s+1, s) = 1$. Analogously, in the permutation function 2, we updated $V(s)$ after $N-1$ steps of updating $V(s+1)$. So, $d_{\phi_2}(s+1.s) = N-1$.

Clearly, the expected number of steps taken by Algorithm 3 $\leqslant CN * \mathbb{E}_\phi[d_\phi(s+1, s)]$ where $C$ is the aforementioned constant in 4(a) and $\mathbb{E}_\Phi[d_\phi(s+1, s)]$ is the expected forward distance between some state $s+1$ and $s$ over all possible permutation functions $\Phi$.

Let's fix some state $s+1$, so we have in total $(N-1)!$ possible permutations. Note that $s$ can be 1 forward distance away, or 2 forward distance away, ... $N-1$ forward distance away from state $s$. Here, for each instance, we are fixing $s+1$ and $s$, thus there are $(N-2)!$ possible permutations. Thus,

$$\begin{aligned}
\mathbb{E}_{\Phi}[d_{\phi}(s+1,s)] &= \frac{1(N-2)! + 2(N-2)! + \cdots + (N-1)(N-2)!}{(N-1)!} \\
&= \frac{\frac{(N-1)N}{2}}{N-1} \\
&= \frac{N}{2} \\
&\leqslant N
\end{aligned}$$

Hence, under an (unknown) arbitrary permutation function and for some given arbitrary initialization, the expected no. of steps taken by Algorithm 3 is **upper bounded by $\mathbf{CN^2}$** for some constant $C$.

□

**Note:** Do not worry about exact constants/one-off differences, as long as the asymptotic solution is correct with the right reasoning, full marks will be given.

5. (5 marks) [TD, MC] Suppose that the system that you are trying to learn about (estimation or control) is not perfectly Markov. Comment on the suitability of using different solution approaches for such a task, namely, Temporal Difference learning, Monte Carlo methods. Explicitly state any assumptions that you are making.

**Solution:**

Recall the definition of $V_\pi(s)$, this will be useful in explanation. For some state $s \in \mathcal{S}$

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

When dealing with a system that is not perfectly Markovian, the Monte Carlo method offers relatively better solutions as compared to Temporal Difference Learning. We have discussed the reasons below:

1. **Temporal Difference (TD) Learning:** TD learning methods, update value estimates based on the difference between temporally separated estimates. TD-$\lambda$ combines elements of both TD(0) and Monte Carlo methods by using eligibility traces, allowing for credit assignment over multiple time steps. The update rule resembles as following:

$$V_\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')]$$

In the context of a non-Markovian system, TD learning is not suitable because of the following reasons:

(a) *Dynamic Programming Updates:* TD methods rely on the Markov property, which assumes that the current state encapsulates all relevant information for prediction or control. In non-Markovian systems, where the current state alone may not provide sufficient information due to dependencies on past states or events, TD updates may not accurately reflect the system's behavior.

(b) *Limited Temporal Information:* TD methods typically update value estimates based on temporal difference errors, which only consider $k$-step-ahead (that may or may not be weighted) predictions (usually $k \approx 1$). In non-Markovian systems with long-term dependencies, where future states and rewards may depend on distant past events, TD updates may fail to capture these dependencies effectively.

(c) *Difficulty in reward Assignment:* Non-Markovian systems may exhibit complex dynamics with delayed effects or hidden states, making it challenging for TD methods to assign rewards accurately. TD updates may incorrectly attribute rewards to states that are not causally related to the observed outcomes.

2. **Monte Carlo Methods:** Monte Carlo methods estimate value functions by averaging returns observed from complete episodes. They do not rely on Markovian assumptions and directly learn from sampled trajectories. In average-Monte Carlo, the value function over $N$ samples is given as follows:

$$V_\pi(s) \leftarrow \frac{1}{N}\sum_{i=1}^{N} G_t(s)$$

For non-Markovian systems, Monte Carlo methods have several advantages:

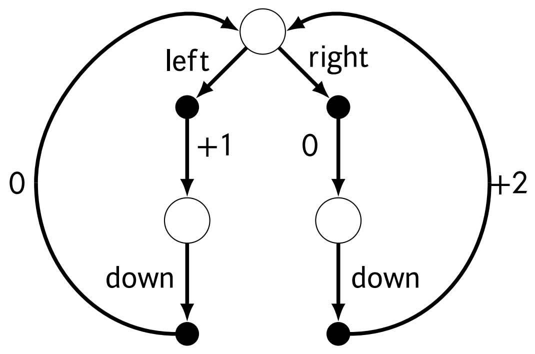
(a) *No Markovian Assumption:* Monte Carlo methods make no assumptions about the Markov property, making them inherently suitable for non-Markovian systems where the current state may not fully encapsulate all relevant information for prediction or control.

(b) *Long-Term Dependencies:* Monte Carlo methods naturally capture long-term dependencies in the system dynamics as they evaluate entire trajectories, with sequences of states and rewards, without relying on temporal difference updates.

(c) *Sample Efficiency:* In some cases, Monte Carlo methods can be more sample efficient than TD learning, especially when trajectories are relatively short or when the Markovian approximation is challenging.

In summary, **Monte Carlo methods offer advantages over TD learning in non-Markovian systems** by directly learning from complete episodes without relying on the Markov property or memoryless updates. They can capture long-term dependencies, complex dynamics, and delayed effects more effectively, making them a preferable choice for learning in such environments.

6. (6 marks) [MDP] Consider the continuing MDP shown below. The only decision to be made is that in the *top* state (say, $s_0$), where two actions are available, *left* and *right*. The numbers show the rewards that are received deterministically after each action. There are exactly two deterministic policies, $\pi_{left}$ and $\pi_{right}$. Calculate and show which policy will be the optimal:



**Solution:** Let the states be denoted as follows:

1. topmost represented state: $\mathsf{s}_0$;
2. bottom-left state: $\mathsf{s}_1$ and
3. bottom-right state $\mathsf{s}_2$.

We denote the state space as $\mathcal{S}$. At $\mathsf{s}_0$ we have two possible actions, namely: $\mathsf{left}$ and $\mathsf{right}$. At each of $\mathsf{s}_1$ and $\mathsf{s}_2$, we have a single possible action, let's call each of them $\mathsf{down}$. We denote the Action space as $\mathcal{A}$. We have two deterministic policies:

1. Going left at state $\mathsf{s}_0$:

$$\pi_{left}(a|s) = \begin{cases} 1 & \text{if } s = \mathsf{s}_0 \text{ and } a = \mathsf{left} \\ 1 & \text{if } s = \mathsf{s}_1 \text{ and } a = \mathsf{down} \\ 1 & \text{if } s = \mathsf{s}_2 \text{ and } a = \mathsf{down} \\ 0 & \text{otherwise} \end{cases}$$

2. Going right at state $\mathsf{s}_0$:

$$\pi_{right}(a|s) = \begin{cases} 1 & \text{if } s = \mathsf{s}_0 \text{ and } a = \mathsf{right} \\ 1 & \text{if } s = \mathsf{s}_1 \text{ and } a = \mathsf{down} \\ 1 & \text{if } s = \mathsf{s}_2 \text{ and } a = \mathsf{down} \\ 0 & \text{otherwise} \end{cases}$$

For the policy $\pi_{left}$, the set of bellman equations goes as follows:

$$\begin{aligned} v_{\pi_{left}}(\mathsf{s}_0) &= 1 + \gamma\, v_{\pi_{left}}(\mathsf{s}_1). \\ v_{\pi_{left}}(\mathsf{s}_1) &= \gamma\, v_{\pi_{left}}(\mathsf{s}_0). \\ v_{\pi_{left}}(\mathsf{s}_2) &= 2 + \gamma\, v_{\pi_{left}}(\mathsf{s}_0). \end{aligned}$$

Solving:

$$\begin{aligned} v_{\pi_{left}}(\mathsf{s}_0) &= \frac{1}{1-\gamma^2} \\ v_{\pi_{left}}(\mathsf{s}_1) &= \frac{\gamma}{1-\gamma^2} \\ v_{\pi_{left}}(\mathsf{s}_2) &= \frac{2-2\gamma^2+\gamma}{1-\gamma^2} \end{aligned} \tag{3}$$

Similarly, for the policy $\pi_{right}$, the set of bellman equations are:

$$\begin{aligned} v_{\pi_{right}}(\mathsf{s}_0) &= \gamma\, v_{\pi_{right}}(\mathsf{s}_2). \\ v_{\pi_{right}}(\mathsf{s}_1) &= \gamma\, v_{\pi_{right}}(\mathsf{s}_0). \\ v_{\pi_{right}}(\mathsf{s}_2) &= 2 + \gamma\, v_{\pi_{right}}(\mathsf{s}_0). \end{aligned}$$

Solving the system of linear equations:

$$\begin{aligned} v_{\pi_{right}}(\mathsf{s}_0) &= \frac{2\gamma}{1-\gamma^2} \\ v_{\pi_{right}}(\mathsf{s}_1) &= \frac{2\gamma^2}{1-\gamma^2} \\ v_{\pi_{right}}(\mathsf{s}_2) &= \frac{2}{1-\gamma^2} \end{aligned} \tag{4}$$

(a) (2 marks) if $\gamma = 0$

**Solution:** For $\gamma = 0$,
the value function for the policy $\pi_{left}$ evaluate as follows:

$$\left(v_{\pi_{left}}(\mathsf{s}_0), v_{\pi_{left}}(\mathsf{s}_1), v_{\pi_{left}}(\mathsf{s}_2)\right) = (1, 0, 2)$$

The value function for the policy $\pi_{right}$ evaluate as follows:

$$\left(v_{\pi_{right}}(\mathsf{s}_0), v_{\pi_{right}}(\mathsf{s}_1), v_{\pi_{right}}(\mathsf{s}_2)\right) = (0, 0, 2)$$

Clearly, $\pi_{left}(s) \geqslant \pi_{right}(s)\ \forall s \in \mathcal{S}$.

Thus, $\boldsymbol{\pi}_{\textbf{left}}$ **is the optimal policy** as compared to $\pi_{right}$.

(b) (2 marks) if $\gamma = 0.9$

**Solution:** For $\gamma = 0.9$,
the value function for the policy $\pi_{left}$ evaluate as follows:

$$\left(v_{\pi_{left}}(\mathsf{s}_0), v_{\pi_{left}}(\mathsf{s}_1), v_{\pi_{left}}(\mathsf{s}_2)\right) = (5.26, 4.74, 6.74)$$

The value function for the policy $\pi_{right}$ evaluate as follows:

$$\left(v_{\pi_{right}}(\mathsf{s}_0), v_{\pi_{right}}(\mathsf{s}_1), v_{\pi_{right}}(\mathsf{s}_2)\right) = (9.47, 8.53, 10.53)$$

Clearly, $\pi_{right}(s) \geqslant \pi_{left}(s)\ \forall s \in \mathcal{S}$.

Thus, $\boldsymbol{\pi}_{\textbf{right}}$ **is the optimal policy** as compared to $\pi_{left}$.

(c) (2 marks) if $\gamma = 0.5$

**Solution:** For $\gamma = 0.5$,
the value function for the policy $\pi_{left}$ evaluate as follows:

$$\left(v_{\pi_{left}}(\mathsf{s}_0), v_{\pi_{left}}(\mathsf{s}_1), v_{\pi_{left}}(\mathsf{s}_2)\right) = (1.33, 0.67, 2.67)$$

The value function for the policy $\pi_{right}$ evaluate as follows:

$$\left(v_{\pi_{right}}(\mathsf{s}_0), v_{\pi_{right}}(\mathsf{s}_1), v_{\pi_{right}}(\mathsf{s}_2)\right) = (1.33, 0.67, 2.67)$$

Clearly, $\pi_{right}(s) = \pi_{left}(s)\ \forall s \in \mathcal{S}$.

Thus, **both policies are equally optimal**.

7. (3 marks) Recall the three advanced value-based methods we studied in class: Double DQN, Dueling DQN, Expected SARSA. While solving some RL tasks, you encounter the problems given below. Which advanced value-based method would you use to overcome it and why? Give one or two lines of explanation for 'why'.

**Solution:** The reader may recall the three advanced value-based methods discussed in class.

1. *Dueling DQN:* There are instances where it is not very crucial to estimate the value of each action choice made in states; in other words, there is not much change between a pair of action values. In this context, if we try to learn the $Q-$ function directly, it will make a lot of unnecessary distinctions in the action values. The Dueling DQN consists of two networks:

   (a) The Value Network learns the value function $V(s)$, and

   (b) The Advantage Network concentrates on learning the advantage function $A(s,a)$.

   (c) The Aggregation Layer to combine $V(s)$ and $A(s,a)$.

   Thus, instead of learning $Q(s,a)$ directly, the state value and the advantage function is used to estimate the $Q-$ value.

$$Q(s,a;\theta,\alpha,\beta) \leftarrow V(s;\theta,\beta) + \left( A(s,a;\theta,\alpha) - \underbrace{\frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)}_{\text{Average advantage}} \right) \tag{5}$$

   Here $\theta$ is the common network parameter, $\alpha$ is the advantage stream parameter, and $\beta$ is the value stream parameter.

2. *Double DQN:* Because the future maximum approximated action value in Q-learning is evaluated using the same Q function as in current action selection policy, in noisy environments Q-learning can sometimes overestimate the action values, slowing the learning. In this context, the Double-Q Learning may be extended to Double DQN by

the addition of a replay memory and a target network with making the $Q-$network as a neural network. Here, two more networks are added (as in Double $Q-$Learning), one each for the online network and the target network to prevent overestimation. The target network is used to estimate the value while the online network is used for selecting the action. The update equation is given as follows:
Here primary network and the target network is denoted by $Q_\theta$ and $Q_{\theta'}$ respectively.

Compute the target $Q$ value:

$$Q^*(s_t, a_t) \approx r_t + \gamma Q_\theta\left(s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a')\right) \tag{6}$$

Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$.
Update target network parameter (usign Polyak averaging):

$$\theta' \leftarrow \tau * \theta + (1-\tau) * \theta' \tag{7}$$

Note that here the estimated value of the discounted future is evaluated using a different parameter, that is — $\theta'$ instead of $\theta$, which solves the overestimation issue.

3. *Expected SARSA:* The reader may recall that in $SARSA$, the agent considers a future sample (in TD(0), it is a one-step sample) and updates the action-value function based on the sample generated.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{8}$$

Note that this sample $a_{t+1}$ may introduce undesired variance which may slow down the convergence. Instead of considering a sample, the use of expectation of $Q-$ value under the followed policy $\pi$ gives us the Expected SARSA update rule as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \mathbb{E}_\pi\left[Q(s_{t+1}, a_{t+1}|s_t\right] - Q(s_t, a_t)\right) \tag{9}$$

Also, note that it is an on-policy update rule. It is because even though a sample of the next step is not drawn as per the policy, the expectation value is considered as per the policy that is followed.

(a) (1 mark) Problem 1: In most states of the environment, choice of action doesn't matter.

**Solution:** Observe that each of Double DQN and Expected SARSA are based on learning through the updation of $Q(s, a)-$value, aka the action-value directly. So, in a setting, where the choice of action doesn't matter, learning $Q(s,a)$ instead of $V(s)$ is an unnecessary suboptimal strategy.

On the other hand, Dueling-DQN separates the value functions into two streams, one for the value of the state $V(s)$ and another for the advantages of each action $A(s,a)$. Explicitly separating the two estimators, the Dueling architecture can learn which states are valuable (or analogously not valuable), without having to learn the effect of each action for each state.

Thus, by decoupling the value estimation, Dueling DQN can effectively handle the situations where the choice of action seems irrelevant in many states, enabling more efficient learning by focusing on the effective learning of the state value in contrast with the action value. Also, since the choices of the actions do not matter, Equation 5 may be reduced as follows:

$$Q(s,a;\theta,\alpha,\beta) \leftarrow V(s;\theta,\beta) + \cancelto{0}{\left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)}$$

Thus, in most of the cases, we only need to worry about learning the $V(s)-$function as compared to learning $A(s,a)$ function. Hence, this disentanglement of $V(s)$ and $A(s,a)$ ensures faster learning.

Conclusively, **Dueling DQN method** should be used in order to capture state values, instead of action-values in the aforementioned setting.

(b) (1 mark) Problem 2: Agent seems to be consistently picking sub-optimal actions during exploitation.

**Solution:** If an agent is constantly picking suboptimal actions, Double DQN can be more effective compared to Dueling DQN or Expected SARSA due to its specific advantages.

Double DQN addresses the overestimation issue present in traditional Q-learning methods, including Dueling DQN and Expected SARSA. Q-learning methods tend to overestimate the values of actions, especially in scenarios where suboptimal actions are frequently chosen. Double DQN mitigates this by decoupling the selection of the action from the evaluation of that action, leading to more accurate action-value estimates.

Double DQN stabilizes learning by using two separate networks to decouple the selection of the best action (using the online network) from the evaluation of that action (using the target network). This helps to prevent the overestimation of action values, making the learning process more reliable and less prone to being misled by suboptimal actions.

While Dueling DQN and Expected SARSA have their own advantages in certain scenarios, **Double DQN** specifically targets the overestimation issue inherent in the Q-learning methods, making it particularly suitable for environments where suboptimal actions are frequently chosen.

(c) (1 mark) Problem 3: Environment is stochastic with high negative reward and low positive reward, like in cliff-walking.

**Solution:** The environment is stochastic, thus any sample chosen for the immediate next action is noisy due to stochasticity, which will introduce unnecessary variance, which may slow down the convergence. On the other hand, the use of the expectation of $Q$ value, aka $\mathbb{E}_{\pi}\left[Q(s_{t+1}, a_{t+1}|s_t\right]$, reduces that variance in the update. Also in stochastic environments, this can be more accurate than Q-learning algorithms, which only consider the maximum Q-value and don't account for the probabilities of different actions. Also, Expected SARSA naturally encourages exploration as it considers all possible actions and their probabilities. This can be beneficial for the agent in stochastic environments where exploration is crucial for discovering the true dynamics of the environment and finding optimal policies.

Overall, in stochastic environments where uncertainty plays a significant role, **Expected SARSA**'s ability to incorporate this uncertainty through its expected value calculation tends to make it more robust and effective for learning the value functions.

8. (2 marks) [REINFORCE] Recall the update equation for *preference* $H_t(a)$ for all arms.

$$H_{t+1}(a) = \begin{cases} H_t(a) + \alpha\,(R_t - K)\,(1 - \pi_t(a)) & \text{if } a = A_t \\ H_t(a) - \alpha\,(R_t - K)\,\pi_t(a) & \text{if } a \neq A_t \end{cases}$$

where $\pi_t(a) = e^{H_t(a)} / \sum_b e^{H_t(b)}$. Here, the quantity $K$ is chosen to be $\bar{R}_t = \left(\sum_{s=1}^{t-1} R_s\right)/t - 1$ because it empirically works. Provide concise explanations to these following questions. Assume all the rewards are non-negative.

(a) (1 mark) How would the quantities $\{\pi_t(a)\}_{a \in A}$ be affected if $K$ is chosen to be a large positive scalar? Describe the policy it converges to.

**Solution:** Note that here $K$ acts as the baseline which $R_t$ is compared to. Putting a very high positive value for $K$ will result in the following case.

If $K$ is significantly larger than the average reward $\overline{R}_t$ the term $K - R_t$ becomes negative irrespective of the actual reward received.

Let's consider the case when $a = A_t$. In the early stage of learning, when the exploration is more relevant we may assume that $\pi_t(a)$ is not close enough to 1. Thus $\Delta H_t(a)$ will suffer a high decrease.

For an arm $a \neq A_t$, it will enjoy an increase due to the high positive value of $-(R_t - K)$ compensated with $\pi_t(a)$, which is less than 1.

Consequently, the policy $\pi_t(a)$ will **tend to approach a more uniform selection** of actions. This occurs because the algorithm imposes a greater penalty on the selected arm while providing rewards to the unselected arms. Such a mechanism **promotes exploration**, even when certain arms demonstrate superiority over others, by fostering a more uniform policy.

(b) (1 mark) How would the quantities $\{\pi_t(a)\}_{a \in \mathcal{A}}$ be affected if $K$ is chosen to be a small positive scalar? Describe the policy it converges to.

**Solution:** On the other hand, if $K$ is a small positive scalar, the term $(R_t - K)$ will predominantly yield positive values, provided that rewards generally exceed $K$.

Consequently, the selected arm $A_t$ will experience more frequent positive updates to its preference $H_{t+1}(A_t)$, thereby elevating the probability $\pi_t(A_t)$ of its selection in subsequent iterations.

For an unchosen arm $a \neq A_t$, its preferences $H_{t+1}(a)$ will diminish, even though to a lesser extent as $(R_t - K)$ approaches zero.

This results in a policy $\pi_t(a)$ **converging towards a more deterministic distribution**, with probability concentrated on arms yielding higher rewards. Such a shift **promotes exploitation** by directing the policy deterministically towards the better-performing arms.

9. (3 marks) [Delayed Bandit Feedback] Provide pseudocodes for the following MAB problems. Assume all arm rewards are Gaussian.

(a) (1 mark) UCB algorithm for a stochastic MAB setting with arms indexed from 0 to $K - 1$ where $K \in \mathbb{Z}^+$.

**Solution:**
**Assumptions:**

(a) For $0 \leqslant i < K$, the reward of Arm$_i$ follows the distribution $\mathcal{N}(\mu_i, \sigma)$. For some constants $\mu_i, \sigma \in \mathbb{R}$.

(b) $\sigma^2$ is known. Typically, $\sigma^2 \approx 1$.

(c) Time Horizon $(N) >$ No. of arms $(K)$

The reader may recall the definition of the indicator function. Given an arbitrary set $\Omega$ and a set $\Gamma \subseteq \Omega$, the *indicator function of* $\Gamma$ is $\mathbb{1}_A : \Omega \to \{0, 1\}$ is defined as follows:

$$\mathbb{1}_A(\omega) = \begin{cases} 1 & \text{if } \omega \in A \\ 0 & \text{otherwise} \end{cases}$$

**Notations:**

(a) $A_* :=$ Estimated Optimal arm

(b) $I_t :=$ Arm played at time $t$

(c) $X_t :=$ Sample reward observed at time $t$

(d) Total reward till time $t$, aka $R_t := \sum_{s=1}^{t} X_s$

(e) No. of times Arm$_i$ has been played till time $t$, aka $T_i(t) := \sum_{s=1}^{t} \mathbb{1}\{I_s = i\}$

(f) The empirical mean of Arm$_i$ at time $t$, aka $\hat{\mu}_i(t) := \dfrac{\sum_{s=1}^{t} \mathbb{1}\{I_s = i\} X_s}{T_i(t)}$

(g) Upper Confidence Bound of Arm$_i$ at time $t$ under parameter $c$, aka

$$UCB_i(t, c) := \hat{\mu}_i(t) + c\sigma\sqrt{\frac{\log t}{T_i(t)}}$$

Now, with these tools, the Upper Confidence Bound Algorithm goes as follows:

**Algorithm 4 :** Upper Confidence Bound Algorithm

```
1: Input: No. of arms: $K$, Time Horizon: $N$, Variance of arms: $\sigma(\approx 1)$ and UCB parameter: $c$
2: Output: Final cumulative reward $R_N$, Best Estimated arm: Optimal−Arm
3: Initialization:
4: for $i = 0, 1, \dots, K-1$ do
5:     $T_i(0) \leftarrow 0$, $\hat{\mu}_i(0) \leftarrow 0$, $R_0 \leftarrow 0$, $UCB_i(0,c) \leftarrow +\infty$
6: end for
7: for $t = 1, 2, \dots N$ do
8:     $I_t \leftarrow \arg\max_{i=0}^{K-1} \text{UCB}_i(t-1,c)$    ▷ Play the arm with the highest UCB
9:     $R_t \leftarrow R_{t-1} + X_t$    ▷ Update total reward
10:    for $i = 0, 1, \dots K$ do
11:        if $i = I_t$ then
12:            $T_i(t) \leftarrow T_i(t-1) + 1$
13:            $\hat{\mu}_i(t) \leftarrow \frac{\hat{\mu}_i(t-1)T_i(t-1) + X_t}{T_i(t)}$    ▷ Update the empirical mean
14:            $\text{UCB}_i(t,c) \leftarrow \hat{\mu}_i(t) + c\sigma\sqrt{\frac{\log t}{T_i(t)}}$    ▷ Update the UCB
15:        else
16:            $T_i(t) \leftarrow T_i(t-1)$
17:            $\hat{\mu}_i(t) \leftarrow \hat{\mu}_i(t-1)$
18:            $\text{UCB}_i(t,c) \leftarrow \text{UCB}_i(t-1,c)$
19:        end if
20:    end for
21: end for
22: $A_* \leftarrow \arg\max_{i=0}^{K-1} \text{UCB}_i(N,c)$    ▷ Set the arm w/- highest UCB as the estimated optimal arm
23: return $R_N, A_*$    ▷ Return the final cumulative reward and estimated best arm
```

(b) (2 marks) Modify the above algorithm so that it adapts to the setting where the agent observes a feedback tuple instead of a reward at each timestep. The feedback tuple $h_t$ is of the form $(t', r_{t'})$ where $t' \sim \text{Unif}(\max(t-m+1, 1), t)$, $m \in \mathbb{Z}^+$ is a constant, and $r_{t'}$ represents the reward obtained from the arm pulled at timestep $t'$.

**Solution:**
Alongside following the Notation of $A_*$, $I_t$, $X_t$ and $T_i(t)$ in 9(a), we introduce the

following notation:

(a) $m :=$ Length of the feedback window

(b) $H_t :=$ Sample feedback tuple observed at time $t$

(c) $Y_t :=$ The timestamp in the sampled tuple $H_t$. As per the question:

$$Y_t \sim U(\max(t - m + 1, 1), t) \tag{10}$$

(d) The number of times till time $t$, the feedback tuple returned the time stamp, at which Arm$_i$ is pulled: $\Xi_i(t) := \sum_{s=1}^{t} \mathbb{1}\{I_{Y_s} = i\}$

(e) Observed total reward till time $t$, aka $P_t := \sum_{s=1}^{t} X_{Y_s}$

As per the Notation in 9(a), note that $H_t$ is of the form $\langle Y_t, X_{Y_t} \rangle$.

To modify Algorithm 4 in order to adapt the setting in which the agent, instead of observing $X_t$ at time $t$, observes $H_t$, we may introduce the Notation of Observed Empirical mean and Observed UCB, that are generalizations of empirical mean and UCB, defined as follows:

(a) Observed empirical mean of Arm$_i$, at time $t$, aka

$$\hat{\beta}_i(t) := \frac{\sum_{s=1}^{t} \mathbb{1}\{I_{Y_s} = i\} X_{Y_s}}{\Xi_i(t)}$$

(b) Observed Upper Confidence Bound of Arm$_i$ at time $t$ under parameter $c$, aka

$$OUCB_i(t, c) := \hat{\beta}_i(t) + c\sigma\sqrt{\frac{\log t}{\Xi_i(t)}}$$

The Algorithm 4 may be generalized to the concerned setting as follows:

**Algorithm 5** : Observed Upper Confidence Bound Algorithm

```
1: Input: No. of arms: $K$, Time Horizon: $N$, Length of the feedback window: $m$, Variance of arms: $\sigma(\approx 1)$ and UCB parameter: $c$
2: Initialization:
3: for $i = 0, 1, \dots, K-1$ do
4:     $\Xi_i(0) \leftarrow 0$, $\hat{\beta}_i(0) \leftarrow 0$, $P_0 \leftarrow 0$, $OUCB_i(0, c) \leftarrow +\infty$
5: end for
6: for $t = 1, 2, \dots N$ do
7:     $I_t \leftarrow \arg\max_{i=0}^{K-1} \mathrm{OUCB}_i(t-1, c)$    ▷ Play the arm with the highest OUCB
8:     Sample $H_t \langle Y_t, X_{Y_t} \rangle$ as per Distribution 10
9:     $P_t \leftarrow P_{t-1} + X_{Y_t}$    ▷ Update total observed reward
10:    for $i = 0, 1, \dots, K$ do
11:        if $i = I_{Y_t}$ then
12:            $\Xi_i(t) \leftarrow \Xi_i(t-1) + 1$
13:            $\hat{\beta}_i(t) \leftarrow \frac{\hat{\beta}_i(t-1)\Xi_i(t-1) + X_{Y_t}}{\Xi_i(t)}$    ▷ Update the observed empirical mean
14:            $\mathrm{OUCB}_i(t, c) \leftarrow \hat{\beta}_i(t) + c\sigma\sqrt{\frac{\log t}{\Xi_i(t)}}$    ▷ Update the OUCB
15:        else
16:            $\Xi_i(t) \leftarrow \Xi_i(t-1)$
17:            $\hat{\beta}_i(t) \leftarrow \hat{\beta}_i(t-1)$
18:            $\mathrm{OUCB}_i(t, c) \leftarrow \mathrm{OUCB}_i(t-1, c)$
19:        end if
20:    end for
21: end for
22: $A_* \leftarrow \arg\max_{i=0}^{K-1} \mathrm{OUCB}_i(N, c)$    ▷ Set the arm w/- highest OUCB as the estimated optimal arm
23: return $P_N$, $A_*$    ▷ Return the observed cumulative reward and the estimated best arm
```

In fact, if we set $m = 1$, Algorithm 5 becomes precisely Algorithm 4. The reader may verify that $m = 1$ will generate the setting in 9(a), and for the setting in 9(a), $Y_t$ is precisely $t$; thus, $P_t$ becomes $R_t$ and $\Xi_i(t)$ becomes precisely $T_i(t)$. Moreover, the observed empirical mean, aka $\hat{\beta}_i(t)$ and the observed upper confidence bound, aka $OUCB_i(t, c)$, become empirical mean ($\hat{\mu}_i(t)$) and upper confidence bound ($UCB_i(t, c)$) respectively. Consequently, the estimated best arm in Algorithm 5 coincides with the same output in Algorithm 4.

10. (6 marks) [Function Approximation] Your are given an MDP, with states $s_1$, $s_2$, $s_3$ and actions $a_1$ and $a_2$. Suppose the states $s$ are represented by two features, $\Phi_1(s)$ and $\Phi_2(s)$, where $\Phi_1(s_1) = 2$, $\Phi_1(s_2) = 4$, $\Phi_1(s_3) = 2$, $\Phi_2(s_1) = -1$, $\Phi_2(s_2) = 0$ and $\Phi_2(s_3) = 3$.

   (a) (3 marks) What class of state value functions can be represented using only these features in a linear function approximator? Explain your answer.

   **Solution:** In a linear function approximator for state value functions, the value of a state is denoted by a sum of weights multiplied by its corresponding features. The linear function approximation can only represent value functions that are expressible as linear combinations of features that are learnable.

   For the given features $\Phi_1(s)$ and $\Phi_2(s)$, which are constants, the class of state value functions that can be represented is **the set of all possible linear combinations of these features $\mathbf{\Phi_i(s_j)}$ for $\mathbf{i \in \{1, 2\}}$ and $\mathbf{j \in \{1, 2, 3\}}$.**

   The state value function $V(s)$ can be represented as follows:

   $$V(s) \approx \omega_o + \omega_1\Phi_1(s) + \omega_2\Phi_2(s).$$

   where $\omega_o$ is the bias term, whereas $\omega_1$ and $\omega_2$ is the weights corresponding to $\Phi_1$ and $\Phi_2$ respectively.

   (b) (3 marks) Updated parameter weights using gradient descent TD(0) for experience given by: $s_2, a_1, -5, s_1$. Assume state-value function is approximated using linear function with initial parameters weights set to zero and learning rate 0.1.

   **Solution:** The state value function $V : S \to \mathbb{R}$ is defined as follows:

   $$V(s) \approx \omega_o + \omega_1\Phi_1(s) + \omega_2\Phi_2(s)$$

   The TD(0) update rule comes as follows:

   $$\omega_i \leftarrow \begin{cases} \omega_i + \alpha(r + \gamma V(s') - V(s)) * 1 \text{ if } i = 0 \\ \omega_i + \alpha(r + \gamma V(s') - V(s))\Phi_i(s) \text{ if } i \neq 0 \end{cases}$$

   Initial weights $w_i = 0$ for $i \in \{0, 1, 2\}$. Thus, $V(s_i) = 0$ for $i \in \{1, 2, 3\}$. Given path traced/ experienced: $s_2, a_1, -5, s_1$, we have:

$$
\begin{aligned}
w &\leftarrow w + \alpha\Phi_i(s)(r + \gamma V(s') - V(s))\Phi_i(s)\\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + 0.1\,(-5 + 1 * 0 - 0)\begin{pmatrix} 1 \\ 4 \\ 0 \end{pmatrix}\\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - 0.5\begin{pmatrix} 1 \\ 4 \\ 0 \end{pmatrix}\\
&= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 2 \\ 0 \end{pmatrix}\\
&= \begin{pmatrix} -0.5 \\ -2 \\ 0 \end{pmatrix}
\end{aligned}
$$

Thus, $\omega \leftarrow \begin{pmatrix} -0.5 \\ -2 \\ 0 \end{pmatrix}$ and $V(s) = -0.5 - 2\Phi_1(s)$ after one step of TD(0) update.