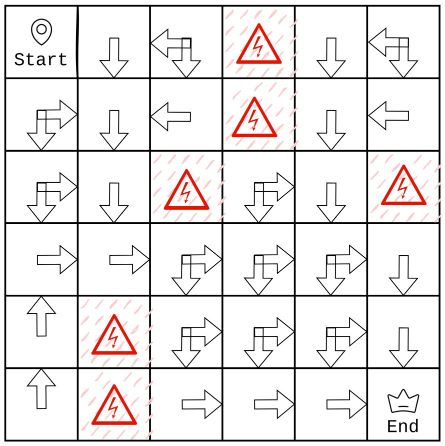
CS6700 : Reinforcement Learning
Written Assignment #1

Topics: Intro, Bandits, MDP, Q-learning, SARSA, FA, DQN **Deadline**: 20/03/2024, 23:55

Name: Girish Madhavan V **Roll number:** ME20B072

- 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.**

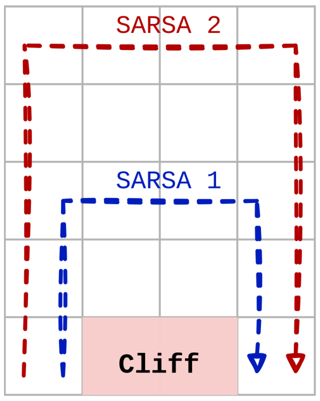
1. (3 marks) [TD, IS] Consider the following deterministic grid-world.



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:
No because following the optimal policy wouldn't allow the agent to explore the states that are not a part of the optimal policy. For example it would impossible for the agent to reach the top right corner while following the optimal policy, hence it would be impossible to learn the value function associated with these states.

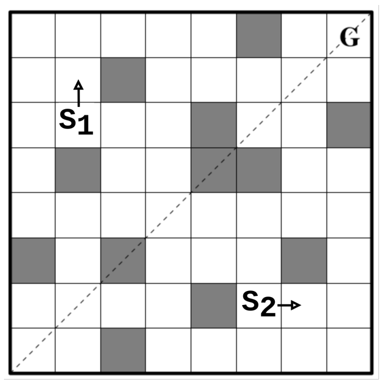
2. (1 mark) [SARSA] In a 5 x 3 cliff-world two versions of SARSA are trained until convergence. The sole distinction between them lies in the ϵ value utilized in their ϵ-greedy policies. Analyze the acquired optimal paths for each variant and provide a comparison of their ϵ values, providing a justification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



Solution:

* The optimal path taken by SARSA 2 tends to be longer than SARSA 1
* This happens due to the more explorative nature for the action selection policy followed in SARSA 2
* From the above we can conclude that the ϵ value of ϵ-greedy policy is larger for SARSA 2 as compared to SARSA 1

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

```
Initialize $Q$-values for all state-action pairs arbitrarily
for each episode do
    Initialize state $s$
    Choose action $a$ using $\epsilon$-greedy policy based on $Q$-values
    while not terminal state do
        Take action $a$, observe reward $r$ and new state $s'$
        Choose action $a'$ using $\epsilon$-greedy policy based on $Q$-values for state $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma Q(s',a') - Q(s,a))$
        $s \leftarrow s'$, $a \leftarrow a'$
    end while
end for
```

Solution:

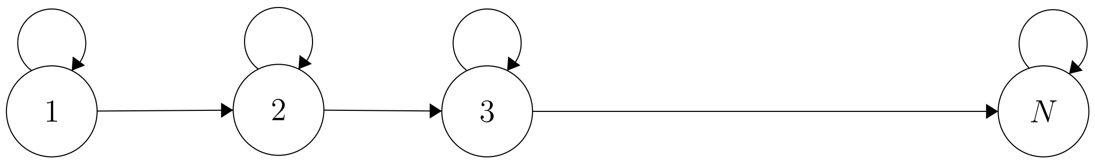
Algorithm 2 SARSA Algorithm

```
Initialize $Q$-values for all state-action pairs arbitrarily
for each episode do
    Initialize state $s$
    Choose action $a$ using $\epsilon$-greedy policy based on $Q$-values
    while not terminal state do
        Take action $a$, observe reward $r$ and new state $s'$
        Choose action $a'$ using $\epsilon$-greedy policy based on $Q$-values for state $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma Q(s',a') - Q(s,a))$
        $\mathbf{Q}(\mathbf{F}(\mathbf{s},\mathbf{a})) \leftarrow \mathbf{Q}(\mathbf{s},\mathbf{a})$
        $s \leftarrow s'$, $a \leftarrow a'$
    end while
end for
```

The modified code allows updating the Q values of the bottom half of the environment with the top half of the environment and vice versa, this makes the algorithm faster as we need to compute the optimal Q values only for half of the states as compared to **Algorithm 1**

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
```

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

Solution: $\phi(k) = N - k + 1$
assuming that all the state values are initialized such that $V(i) =$ large negative value whose magnitude is greater than N $\forall i = 1..N-1$ and $V(N) = 0$, the number of steps taken to convergence is **N** steps as every state value function would converge in one step. This initialization would lead to the fastest convergence.
1st iteration : $V(N) = 0 + \gamma \cdot 0 = 0$
2nd iteration : $V(N-1) = \max(0 + \gamma \cdot V(N), -1 + \gamma \cdot V(N-1)) = 0$
3rd iteration : $V(N-2) = \max(-1 + \gamma \cdot V(N-1), -1 + \gamma \cdot V(N-2)) = -1$
4th iteration : $V(N-3) = \max(-1 + \gamma \cdot V(N-2), -1 + \gamma \cdot V(N-3)) = -1 - \gamma$
...
ith iteration : $V(i) = \frac{\gamma^{i-N+1} - 1}{1 - \gamma}$

The optimal Value of $V(N)$ is 0 because, when $V(N)$ is initialized as any non zero value after k iterations $V^k(N) = \gamma^k V^0(N)$ [the only action possible at terminal state is to stay there] as k increases $\gamma^k \to 0$ as $\gamma < 1$. Hence $V^*(N) = 0$, also it logical to set the state value function as zero in the terminal state as there is no possibility collecting rewards from the terminal state. Since we are initializing the terminal state value function with the optimal value, all the other state's value functions computed from them also end up giving their optimal values.

2. (1 mark) Design a permutation function ϕ 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: $\phi(k) = k$
1st iteration : $V(1) = \max(-1 + \gamma \cdot V(1), -1 + \gamma \cdot V(2)) \rightarrow$ not converged
2nd iteration : $V(2) = \max(-1 + \gamma \cdot V(2), -1 + \gamma \cdot V(3)) \rightarrow$ not converged
...
Nth iteration : $V(N) = 0 + \gamma \cdot V(N) = 0$
...
N + N - 1 the iteration : $V(N-1) = \max(0 + \gamma \cdot V(N), -1 + V(N-1)) = 0$
...
So this permutation function allows the code to converge in N^2 iterations as only one state's value function converges in N iterations.

3. (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: From the previous problems we can say that to find the optimal values for all states the permutation must gives the states as $N, N-1, N-2, ...1$. So the expected number of iterations for all the states to converge is $\mathbb{E}[X_N] + \mathbb{E}[X_{N-1}] + ... + \mathbb{E}[X_1]$ where X_i is the random variable denoting the number of times the permutation function needs to be invoked to get the number i as output.

$\mathbb{E}[X_i] = \frac{1}{N} + \frac{2\cdot(N-1)}{N^2} + \frac{3\cdot(N-1)^2}{N^3} + ... + \frac{j\cdot(N-1)^{j-1}}{N^j} .....j \rightarrow \infty$

$\mathbb{E}[X_i] = \Sigma_{j=1}^{\infty} \frac{j\cdot(N-1)^{j-1}}{N^j}$

$\mathbb{E}[X_i] = \frac{1}{N-1} \Sigma_{j=1}^{\infty} j \cdot x^j$,where $x = \frac{N-1}{N}$

$x \cdot \mathbb{E}[X_i] = \frac{1}{N-1}\Sigma_{j=1}^{\infty} x^{j+1}$

Now on subtracting both of them we get :

$\mathbb{E}[X_i] = \frac{1}{N-1} \cdot \frac{x}{(1-x)^2} = N$

Hence the total expected number of steps taken to converge is
$N + N + ...(N \text{ times}) = N^2$, which is of the **Order** N^2

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.

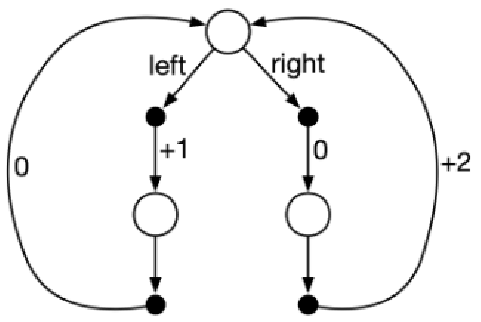
5. (5 marks) [TD, MC] Suppose that the system that you are trying to learn about (estimation or control) is not perfectly Markov. Comment on the suitability of using different solution approaches for such a task, namely, Temporal Difference learning, Monte Carlo methods. Explicitly state any assumptions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

Solution:
Temporal Difference Learning : $V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$
Here the term $r_{t+1} + \gamma V(s_{t+1})$ is the estimate of the return G_t and computing this estimate requires the assumption of Markov Decision Process, Hence Temporal difference learning might perform badly in an environment that is not perfectly Markov.

Monte Carlo method : $V(s_t) \leftarrow V(s_t) + \alpha \cdot [G_t - V(s_t)]$ This would calculate the Value function by starting from an arbitary starting state and following the policy till reaching the terminal state, once the episode is completed the return for the states is computed using the discount factor and then the average return for that state is stored in state value function. When this simulation is repeated several times the state value functions can be expected to converge to their values. Since we haven't assumed any markov property for the environment this method tends to give better results in an environment that is not perfectly Markov.

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, π_{left} and π_{right}. Calculate and show which policy will be the optimal:

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



Solution:
Bellman optimality equation for deterministic environment :

$$V^*_\pi(s) = \max_a r + \gamma V^*_\pi(s')$$

Using this we can start computing the optimal state values as follows :

$V^*_{\pi_{left}}(s_{1,left}) = 0 + \gamma V^*_{\pi_{left}}(s_0) = \gamma V^*_{\pi_{left}}(s_0)$
$V^*_{\pi_{left}}(s_0) = 1 + \gamma V^*_{\pi_{left}}(s_{1,left})$
$V^*_{\pi_{left}}(s_0) = 1 + \gamma^2 V^*_{\pi_{left}}(s_0)$
$V^*_{\pi_{left}}(s_0) = 1/(1-\gamma^2)$

$V^*_{\pi_{right}}(s_{1,right}) = 2 + \gamma V^*_{\pi_{right}}(s_0)$
$V^*_{\pi_{right}}(s_0) = 0 + \gamma V^*_{\pi_{right}}(s_{1,right})) = \gamma V^*_{\pi_{right}}(s_{1,right})$
$V^*_{\pi_{right}}(s_0) = \gamma(2 + \gamma V^*_{\pi_{right}}(s_0))$
$V^*_{\pi_{right}}(s_0) = 2\gamma/(1-\gamma^2)$

So when $\gamma = 0$, $V^*_{\pi_{left}}(s_0) = 1$ and $V^*_{\pi_{right}}(s_0) = 0$
As $V^*_{\pi_{left}}(s_0) > V^*_{\pi_{right}}(s_0)$, we can say that π_{left} is better than π_{right}

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

Solution:
Similarly, substituting $\gamma = 0.9$ we have

$$V^*_{\pi_{left}}(s_0) = 5.263 \text{ and } V^*_{\pi_{right}}(s_0) = 9.473$$

As $V^*_{\pi_{left}}(s_0) < V^*_{\pi_{right}}(s_0)$ we can say that π_{right} is better than π_{left}

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

Solution:
Similarly, substituting $\gamma = 0.5$ we have

$$V^*_{\pi_{left}}(s_0) = 1.333 \text{ and } V^*_{\pi_{right}}(s_0) = 1.333$$

As $V^*_{\pi_{left}}(s_0) = V^*_{\pi_{right}}(s_0)$ we can say that π_{right} and π_{left} are equally good

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'.

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

   Solution: Dueling DQN, as this works by splitting the state action function [Q] into state value [V] and Advantage function. When the actions are equally good, there wouldn't be any reason to prefer any particular action in a given state, doing so would make the agent jitter around a lot with no meaningful gain in reward for the action as compared to following the previous action.

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

   Solution: Double DQN, because of the presence of maximization bias would skew the Temporal Difference error and this can be minimized by using different Q values for computing the action and the next state action value

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

   Solution: Expected SARSA, because this uses the state value function instead of the state action function in the temporal difference error. Since the state value function is an expectation of the state action function, this would help minimize the noise/variance in our model introduced due to its stochastic nature

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

Solution:
In this case since the baseline [K] is larger than the mean reward of the bandit system, we can expect the term $R_t - K$ to be negative most of the time as the chances of R_t being more than K is fewer. This would cause a decreased in preference for the action taken at time(t) and and increase in preferences for all the remaining actions, such a change in $H(a)$ is reflected in $\pi_t(a)$ as decrease in numerator for action(a) taken at time(t) while increase in numerator for remaining actions. Such a policy of levying penalty for good actions and rewarding other actions is characteristic of an explorative 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: In this case the term $R_t - K$ tends to be mostly positive as K is less than the mean reward, this leads to increase in perference for chosen action(a) at time(t) that gives reward more than K while other actions have reduced preference. This change in $H(a)$ affects $\pi_t(a)$ by increasing the numerator for action(a) taken at time(t) that has generated a reward more than K while other actions are penalised with a lower preference. Such a policy that chooses the first best action several times is characteristic to that of an exploitative policy.

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:

Algorithm 1 UCB, Stochastic Multi Armed Bandit

```
Play every machine once
Initialize
    total_reward[j] ← reward obtained from using machine j
    number_pulls ← [1 1 ..... 1]
    timestep ← 1
for each timestep < Total time do
    Initialize mean_reward[0...K-1] ← $\frac{total\_reward[0...K-1]}{number\_pulls[0...K-1]}$
    Initialize UCB[0...K-1] ← mean_reward[0...K-1] + $\sqrt{\frac{2\cdot\ln(timestep)}{number\_pulls[0...K-1]}}$
    Choose Best Arm ← $\arg\max_{j\in[0...K-1]} UCB[j]$
    Update
        timestep ← timestep + 1
        number_pulls[Arm] ← number_pulls[Arm] + 1
        total_reward[Arm] ← total_reward[Arm] + $r_t$
end for
```

(b) (2 marks) Modify the above algorithm so that it adapts to the setting where agent observes a feedback tuple instead of 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:

Algorithm 2 UCB, Delayed Bandit Feedback

```
Play every machine once
Initialize
    total_reward[j] ← reward obtained from using machine j
    number_pulls ← [1 1 ..... 1]
    timestep ← 1
    m ← Input()
    Delay[m]
for each timestep < Total time do
    Initialize mean_reward[0...K-1] ← $\frac{total\_reward[0...K-1]}{number\_pulls[0...K-1]}$
    Initialize UCB[0...K-1] ← mean_reward[0...K-1] + $\sqrt{\frac{2 \cdot \ln(timestep)}{number\_pulls[0...K-1]}}$
    Choose Best Arm ← $\arg\max_{j \in [0...K-1]} UCB[j]$
    Update
        Generate t' ∼ Unif(max(timestep-m+1,1),timestep)
        number_pulls[Arm] ← number_pulls[Arm] + 1
        $r_{t'}$ ← Delay[(t'-timestep)mod(m)]
        total_reward[Arm] ← total_reward[Arm] + $r_{t'}$
        if timestep < m then
            Delay[timestep] ← (timestep, $r_t$)
        else
            Copy Delay[1...m-1] into Delay [0...m-2]
            Delay[-1] ← (timestep, $r_t$)
        end if
        timestep ← timestep + 1
end for
```

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 $\Phi_2(s_3) = 3$.

    1. (3 marks) What class of state value functions can be represented using only these features in a linear function approximator? Explain your answer.

Solution:

$$V(s, \mathbf{w}) = w_1 \cdot \Phi_1(s) + w_2 \cdot \Phi_2(s) + w_0$$

This form represents the class of state value functions that can be obtained by combining $\Phi_1(s)$ and $\Phi_2(s)$ in a linear combination with their weights w_1 and

w_2 along with a bias term. Since the function is a linear combination of the features it belongs to a family of linear functions approximators.

2. (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: Given, $s_2, a_1, R = -5, s_1$, $\alpha = 0.1$, Initialize weights to 0
Update rule for gradient descent TD(0),

$$V(s, \mathbf{w}) = w_0 + w_1\Phi_1(s) + w_2\Phi_2(s)$$
$$\delta_t = r_{t+1} + \gamma \max_a V(s_{t+1}, \mathbf{w^t}) - V(s_t, \mathbf{w^t})$$
$$\nabla_{w^t}[r_{t+1} + \gamma \max_a V(s_{t+1}, \mathbf{w^t}) - V(s_t, \mathbf{w^t})]^2 = -2\delta_t \Phi(s_t)$$
$$\mathbf{w^{t+1}} = \mathbf{w^t} + \alpha\, \delta_\mathbf{t}\, \mathbf{\Phi(s_t)}$$

Using the above equation we can derive the update rule and setting initial weights $\mathbf{w^0}$ as [0 0 0]' as asked in the question, we get

$V(s1, \mathbf{w^0}) = 0$ and $V(s2, \mathbf{w^0}) = 0$

$$w_0^1 = w_0^0 + \alpha \cdot (r_1 + \gamma \cdot V(s_1, \mathbf{w^0}) - V(s_2, \mathbf{w^0})) \cdot 1$$
$$w_0^1 = 0 + 0.1 \cdot (-5 + \gamma \cdot 0 + 0) \cdot 1 = -0.5$$

$$w_1^1 = w_1^0 + \alpha \cdot (r_1 + \gamma \cdot V(s_1, \mathbf{w^0}) - V(s_2, \mathbf{w^0})) \cdot \Phi_1(s_2)$$
$$w_1^1 = 0 + 0.1 \cdot (-5 + \gamma \cdot 0 + 0) \cdot 4 = -2$$

$$w_2^1 = w_2^0 + \alpha \cdot (r_1 + \gamma \cdot V(s_1, \mathbf{w^0}) - V(s_2, \mathbf{w^0})) \cdot \Phi_2(s_2)$$
$$w_2^1 = 0 + 0.1 \cdot (-5 + \gamma \cdot 0 + 0) \cdot 0 = 0$$

The updated weights are : $w_1 = -2, w_2 = 0, w_0 = -0.5$