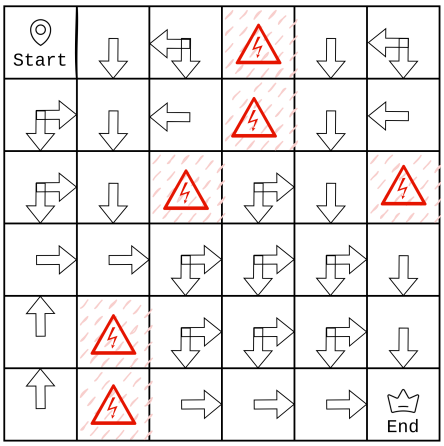
CS6700 : Reinforcement Learning
Written Assignment #1

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

Name: Srikar Babu Gadipudi **Roll number:** EE21B138

- 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.

Learning a policy (value function for the policy) from playing another policy is known as off-policy learning. To learn a policy in an off-policy fashion the behavioral policy (policy being played) needs to encompass all the state-action pairs of the policy to be learned. In detail, to estimate value function values for any policy using behavior policy we utilize the importance sampling ratio

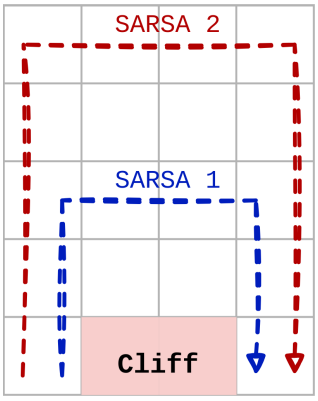
$$\rho_t^T = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \tag{1}$$

where π is the policy to be learned and μ is the behavior policy. This ratio is multiplied with the return to estimate the value function of the policy to be learned. And the ratio is well defined only when $\mu(A_k|S_k)$ is non-zero for every non-zero value of $\pi(A_k|S_k)$.

In this case, the behavior policy is the optimal policy and the policy to be learned is any arbitrary policy. We can clearly observe that the optimal policy does not consider all state-action pairs that an arbitrary policy might follow.

For example, let the policy that needs to be learned take the action 'down' from all states. The value function for all states cannot be estimated until we play that action ('down') from every state. This is not the case when we follow the optimal policy in some states, as seen from the Figure 1 ($\mu(A_k|S_k)$ is zero for some non-zero $\pi(A_k|S_k)$).

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: SARSA 2 will have a higher ϵ value compared to SARSA 1.

We know that SARSA is an on-policy algorithm. SARSA uses the next state-action pair values to estimate the current value function.

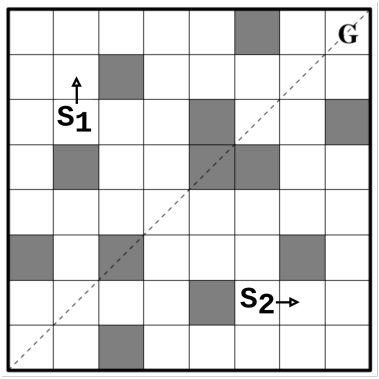
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{2}$$

Here the actions A_t and A_{t+1} are taken from states S_t and S_{t+1} respectively following the ϵ greedy policy. SARSA algorithm estimates the action-value functions as an estimation of the return following the ϵ greedy policy.

A higher ϵ value implies more exploration. In the environment provided above, falling down the cliff would result in a high negative reward. Hence, the action-value function for the states near the cliff will be lower with higher exploration. Therefore, SARSA algorithm with higher exploration tends to choose a path that is farther away from the cliff.

We observe that SARSA 2 takes a longer route to avoid the cliff. Thus, we can imply that SARSA 2 explores more than SARSA 1 algorithm. Therefore SARSA 2 has a higher ϵ value when 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 ϵ-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 ϵ-greedy policy based on Q-values for state s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s'$, $a \leftarrow a'$
  end while
end for

Solution: We can leverage the symmetry present in the environment by simultaneously updating the action-value function for the state-action pair that is symmetrically complimentary to the current state-action pair that is being updated. The updated algorithm will be the following:

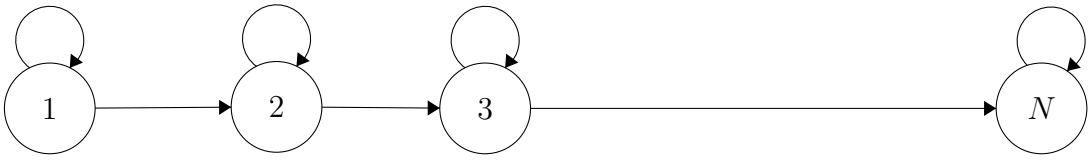
Algorithm 2 SARSA Updated Algorithm

Initialize Q-values for all state-action pairs arbitrarily
for each episode **do**
  Initialize state s
  Choose action a using ϵ-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 ϵ-greedy policy based on Q-values for state s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$
    $Q(F(s,a)) \leftarrow Q(s,a)$
    $s \leftarrow s'$, $a \leftarrow a'$
  end while
end for

This update can be done because symmetry ensures that the value function values are the same for symmetrically opposite/complementary states.

Another way of leveraging symmetry is by reducing the state space of the environment. We can only consider the upper triangle (above the off-diagonal) or lower triangle (below the off-diagonal) alone to be our state space. We can simply copy the action-value functions to the other half as they will converge to the same values anyway due to symmetry. But this will require us to redefine the environment by making corresponding changes to the action space as well.

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

Initialize V-values for all states arbitrarily over the state space S of N states. Define a permutation function ϕ over the state space
$i \leftarrow 0$
while NotOptimal(V) **do**
    $s \leftarrow \phi(i \mod N + 1)$
    $V(s) \leftarrow \max_a \sum_{s,r'} p(s', r|s, a)[r + \gamma * V(s')]$
    $i \leftarrow i + 1$
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: The fastest convergence is obtained when we choose, considering ϕ maps from state space to itself

$$\phi(x) = N - x + 1 \tag{3}$$

This is essentially a permutation function that ensures that the updates happen from the last state/terminal state to the first state. The number of steps to converge to the optimal value function values would be $N - 1$ when the value function values for all states are initialized to '-N' or less (except for the terminal state which is initialized to '0'). This is of the order - $O(N)$. The number of steps to converge to the optimal value function would be of the order $O(N^2)$ when arbitrary initialization is used (some states having initial values greater than '-N'). But still, this permutation ensures faster convergence to the optimal value function compared to other permutations.

Note: The exact number of steps for convergence depends on the initialization for the value function values chosen.

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: The slowest convergence is obtained when we choose, considering ϕ maps from state space to itself

$$\phi(x) = x \tag{4}$$

This is essentially a permutation function that ensures that the updates happen from the first state to the last state. The number of steps to converge to the optimal value function values would be $(N-1)^2$ steps when the value function values for all states are initialized to be '0'. This is of the order - $O(N^2)$.

Note: The exact number of steps for convergence depends on the initialization for the value function values chosen.

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 working out the value iteration for some random set of states for several iterations, we can observe that the value function for a particular state converges only when the state that follows it has already converged. We can use this to our advantage to compute the expected number of steps following the value iteration algorithm with random state selection. At every step, the probability of choosing a particular state for updating is $\frac{1}{N}$. Following this we can conclude that the expected number of steps to converge to the optimal value function of a particular state given that the next state has already converged is given by

$$\mathbb{E}[\text{No. of steps to converge to } v(s_n)] = \frac{1}{N} + \frac{N-1}{N}\frac{1}{N} * 2 + (\frac{N-1}{N})^2\frac{1}{N} * 3 + \dots \tag{5}$$

This arithmetic geometric progression results in N. The value iteration algorithm ends up repeating this process for $N-1$ states (assuming the terminal

state is initialized with 0). Therefore, the expected number of steps required to converge would be $(N-1)*N$. This is of the order - $O(N^2)$, but the expected number will be lesser than that of the case where updation happens in order (from the first state to the last state). The number of steps taken will be greater than N but lesser than exactly N^2 but closer to the order of 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: The system is not perfectly Markovian, meaning the future state depends not only on the current state and action but also potentially on the past history. We hypothesize Monte Carlo methods are better when compared to Temporal Difference learning in a non-Markovian setting.

- **Temporal Difference Learning**

$$\hat{v}_\pi(s_t) = \hat{v}_\pi(s_t) + \alpha(R_{t+1} + \hat{v}_\pi(s_{t+1}) - \hat{v}_\pi(s_t)) \tag{6}$$

  - TD learning is less suitable in a non-Markovian setting.
  - TD methods learn incrementally using one-step updates. While past history still affects future states, TD can leverage the Markov property of the immediate transition (current state, action, next state, reward). The estimates might be biased due to the mismatch between the single-step update and the non-Markovian nature of the system.
  - Additionally, if we use TD(λ) the influence of past states mey be better considered to some extent, potentially improving performance in non-Markovian settings.

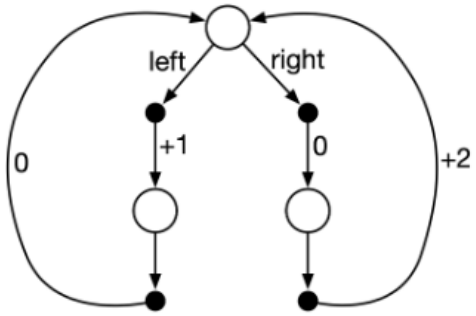
- **Monte Carlo Methods**

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{7}$$

  - Monte Carlo methods are more suitable in a non-Markovian setting.
  - MC methods average returns from complete episodes, which inherently capture the influence of past states on future rewards, even in non-Markovian environments. Although learning can be slow due to high variance, MC methods can still provide unbiased estimates of state values.

While both have their limitations in non-Markovian settings, we expect MC methods to perform better. They can provide unbiased estimates even with the influence of past states, although they might be slow to converge due to variance. TD methods can struggle with bias in strongly non-Markovian 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, π_{left} and π_{right}. Calculate and show which policy will be the optimal:



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

Solution: The optimal action is determined by comparing $Q(s_0, left)$ and $Q(s_0, right)$. For the following deterministic policy

$$v_{\pi_{left}}(s_0) = 1 + \gamma^2 + \gamma^4 + ...\infty \tag{8}$$

$$v_{\pi_{right}}(s_0) = 2 \times (\gamma + \gamma^3 + \gamma^5 + ...\infty) \tag{9}$$

Using $\gamma = 0$, we get $Q(s_0, left) = 1$ and $Q(s_0, right) = 0$. Therefore, the optimal policy will be π_{left}.

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

Solution: The optimal action is determined by comparing $Q(s_0, left)$ and $Q(s_0, right)$. For the following deterministic policy

$$v_{\pi_{left}}(s_0) = 1 + \gamma^2 + \gamma^4 + ...\infty \tag{10}$$

$$v_{\pi_{right}}(s_0) = 2 \times (\gamma + \gamma^3 + \gamma^5 + ...\infty) \tag{11}$$

Using $\gamma = 0.9$, we get $Q(s_0, left) = 5.263$ and $Q(s_0, right) = 9.474$. Therefore, the optimal policy will be π_{right}.

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

Solution: The optimal action is determined by comparing $Q(s_0, left)$ and $Q(s_0, right)$. For the following deterministic policy

$$v_{\pi_{left}}(s_0) = 1 + \gamma^2 + \gamma^4 + ...\infty \tag{12}$$

$$v_{\pi_{right}}(s_0) = 2 \times (\gamma + \gamma^3 + \gamma^5 + ...\infty) \tag{13}$$

Using $\gamma = 0.5$, we get $Q(s_0, left) = 1.333$ and $Q(s_0, right) = 1.333$. Therefore, both π_{left} and π_{right} are optimal polices.

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
   Even with similar rewards, the advantage function used in Dueling DQN architecture helps the agent focus on actions that outperform the average in a state, leading to efficient learning. Also, this will ensure that the action-value function will be equal to the state-value function for the states with similar returns for all actions from that state.

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

   Solution: Double DQN
   It tackles sub-optimal exploitation by separating action selection and evaluation. The main network picks the action with the highest Q-value (exploitation) while the target network estimates Q-values for the chosen action (evaluation, avoids

overestimating from the main network). Essentially solving the maximization bias problem which is the reason for sub-optimal action selection.

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

Solution: Expected SARSA
It tackles situations like cliff-walking by averaging future rewards, encouraging exploration and reducing overestimation of risky actions. Instead of considering only the value function of the next state following the action taken, we consider the expectation over all actions from the next state which reduces variance.

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: When the value of K is chosen is large the policy will converge to the optimal policy as the number of pulls becomes very large. But for practical purposes, the policy might oscillate between optimal and sub-optimal arms for a large number of pulls.

At time t consider the update equation given above. H_{t+1} will decrease for $a = A_t$ and increase for all $a \neq A_t$ (considering $R_t < K$). This leads to a decrease in $\pi_t(a)$ for $a = A_t$ and increase in $\pi_t(a)$ for all $a \neq A_t$. If this continues throughout training then the policy will not be able to identify the best possible action in a sensible number of steps. It results in a policy that oscillates between optimal and sub-optimal actions.

But from the perspective of a baseline not adding any bias to the expected value of the reward, we can say that this policy also converges to the optimal policy but in a large number of steps due to a small incremental update in $H_t(a^*)$ (where is a^* is the optimal arm). Although the above explanation may be true for real-life execution for finite steps, when the number of pulls tends to infinity the policy will converge to the optimal as baseline does not add bias to the

update equation unless it depends on the actions being taken. Here it does not depend on the actions being taken, therefore it will converge to the optimal as the number of pulls tends to infinity.

(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: When the value of K chosen is small the policy will converge to the optimal policy, but this also might take a long time to converge due to variance. At time t consider the update equation given above. H_{t+1} will increase for $a = A_t$ and decrease for all $a \neq A_t$ (considering $R_t > K$) whose values will be proportional to the rewards obtained from taking that action. This leads to a decrease in $\pi_t(a)$ for $a = A_t$ and increase in $\pi_t(a)$ for all $a \neq A_t$. This will eventually converge to the optimal policy but might end up taking time due to high variance in the observations.

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: Here is the UCB algorithm for stochastic MAB setting with K arms.

Algorithm 4 UCB Algorithm

Initialize:
Play each arm once and assign $Q_i \leftarrow R_i$
$n \leftarrow K$
$n_i \leftarrow 1$ (for all $i \in \{0, 1, \ldots, K-1\}$)
c: Exploration constant (real number)
while forever **do**
  Play the arm (i) that maximizes $Q_j + c\sqrt{\frac{\ln(n)}{n_j}}$ where $j \in \{0, 1, \ldots, K-1\}$ and observe the reward obtained (R)
  $Q_i \leftarrow Q_i + \frac{1}{n_i+1}(R - Q_i)$
  $n = n + 1$
  $n_i = n_i + 1$
end while

(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: Here is the UCB algorithm for the delayed feedback scenario.

Algorithm 5 UCB Algorithm Delayed Feedback

Initialize:
Assign $Q_i \leftarrow 0$ (for all $i \in \{0, 1, \ldots, K-1\}$)
$n \leftarrow 0$
$n_i \leftarrow 0$ (for all $i \in \{0, 1, \ldots, K-1\}$)
c: Exploration constant (real number)
Memory $\leftarrow$ Empty List
while forever **do**
  if any arm's feedback (i^{th} arm) has never been observed **then**
    Play arm i, and observe the reward $(t', r_{t'})$
  end if
  if feedback from all arms have been observed at least once **then**
    Play the arm (i) that maximizes $Q_j + c\sqrt{\frac{\ln(n)}{n_j}}$
    where $j \in \{0, 1, \ldots, K-1\}$ and observe the reward obtained $(t', r_{t'})$
  end if
  if $(t', r_{t'})$ is not in Memory **then**
    Let j be the arm played at t'
    $Q_j \leftarrow Q_j + \frac{1}{n_i+1}(R - Q_j)$
    $n = n + 1$
    $n_j = n_j + 1$
    Append $(t', r_{t'})$ to Memory
  end if
end while

The first 'if' condition ensures that all the Q_i values are initialized to the first case of observed rewards. The second 'if' condition ensures we follow the UCB action picking algorithm. And finally the last 'if' condition updates Q_i values only when $(t', r_{t'})$ has never been used before.

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:

$$\phi(s_1) = [2, -1] \tag{14}$$
$$\phi(s_2) = [4, 0] \tag{15}$$
$$\phi(s_3) = [2, 3] \tag{16}$$

The class of state value functions that can be represented using only these features in a linear function approximation are hyperplanes.

A linear function approximation with two features takes the form:

$$V(s) = w_1 * \phi_1(s) + w_2 * \phi_2(s) + b \tag{17}$$

where w_1 and w_2 are weights corresponding to the features ϕ_1 and ϕ_2 and b is the bias term.

By changing the weights w_1 and w_2, this function can represent any plane in the space defined by the two features. This is because varying the weights changes the slope and intercept of the line, essentially creating different hyperplanes.

Therefore, we get

$$V(s_1) = 2w_1 - w_2 + b \tag{18}$$
$$V(s_2) = 4w_1 + b \tag{19}$$
$$V(s_3) = 2w_1 + 3w_2 + b \tag{20}$$

These are the possible state value function equations for the given state space.

Note: To ensure completeness, we know that linear function approximation we can also have different combinations of the features, such as $\phi_1\phi_2$, ϕ_1^2, ϕ_2^2, etc., weighted with linear weights. These sets of functions are also termed linear function approximations as they are linear with respect to the weights and biases that are to be estimated.

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: For the experience $s_2, a_1, -5, s_1$ where the weights are initialized to 0, we can obtain the updated weight values by using the TD(0) update equation,

which is given by

$$w_{t+1} = w_t - \frac{1}{2}\alpha\nabla_{w_t}[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]^2 \tag{21}$$

$$w_{t+1} = [0,0,0] - \frac{1}{2}\alpha\nabla_{w_t}[-5 + \gamma V(s_1) - V(s_2)]^2 \tag{22}$$

$$w_{t+1} = [0,0,0] - \frac{1}{2}\alpha\nabla_{w_t}[-5 + \gamma(2w_1 - w_2 + b) - (4w_1 + b)]^2 \tag{23}$$

where $\alpha = 0.1$ and we assume $\gamma = 1$ (undiscounted returns). Here we can perform the gradient of the TD error using the semi-gradient technique.

$$\nabla_{w_t}[-5 + \gamma(2w_1 - w_2 + b) - (4w_1)]^2 = -10 * [-4, 0, -1] \tag{24}$$

Substituting this in the weight update equation we get

$$w_{t+1} = [0,0,0] - \frac{1}{2}(0.1) * (-10 * [-4, 0, -1]) \tag{25}$$

Solving this we get $w_{t+1} = [-2, 0, -0.5]$.
Note: Using the regular gradient gives us $w_{t+1} = [-1, -0.5, 0]$