1. What are the assumptions required to apply DP in RL?

DYNAMIC PROGRAMMING

- Dynamic Programming is a mathematical optimization approach typically used to improvise recursive algorithms.
- It basically involves simplifying a large problem into smaller sub-problems.
- There are two properties that a problem must exhibit to be solved using dynamic programming:

1. Overlapping Subproblems
2. Optimal Substructure

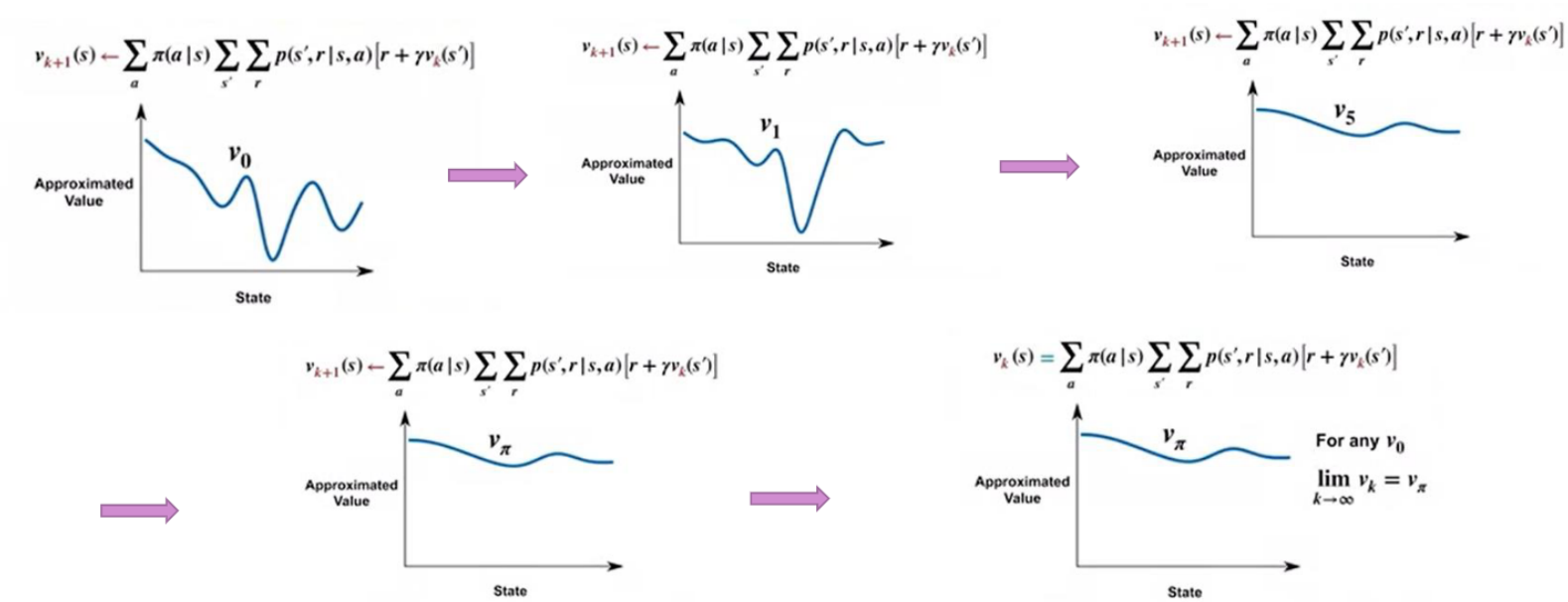
- The term dynamic programming (DP) in RL refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).
- Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically.

- Assume that the environment is a finite MDP.
- That is, we assume that its state, action, and reward sets, S, A(s), and R, for s ∈ S, are finite, and that its dynamics are given by a set of probabilities p(s' , r|s, a), for all s ∈ S, a ∈ A(s), r ∈ R, and s' ∈ S + (S+ is a terminal state if the problem is episodic).
- The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies
- we can easily obtain optimal policies once we have found the optimal value functions, v∗ or q∗, which satisfy the Bellman Optimality Equations

2. Explain the concept of Iterative Policy Evaluation with the help of an appropriate diagram.

ITERATIVE POLICY EVALUATION

- Iterative Policy Evaluation applies the expected updates for each state, iteratively, until the convergence to the true value function is met.
- Let's see visually how this procedure works. We begin with an arbitrary initialization for our approximate value function, let's call this v_0. Each iteration then produces a better approximation by using the update rule shown at the top of the slide. Each iteration applies these updates to every state, S, in the state space, which we call a sweep. Applying this update repeatedly leads to a better and better approximation to the state value function v Pi. If this update leaves the value function approximation unchanged, that is, if v_k plus 1 equals v_k for all states, then v_k equals v Pi, and we have found the value function.
- This is because v Pi is the unique solution to the Bellman equation. The only way the update could leave v_k unchanged is if v_k already obeys the Bellman equation. In fact, it can be proven that for any choice of v_0, v_k will converge to v Pi in the limit as k approaches infinity.



3. Write the pseudocode for Iterative Policy Evaluation.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated

$V \leftarrow \vec{0}, V' \leftarrow \vec{0}$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

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

$$\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$$

$V \leftarrow V'$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

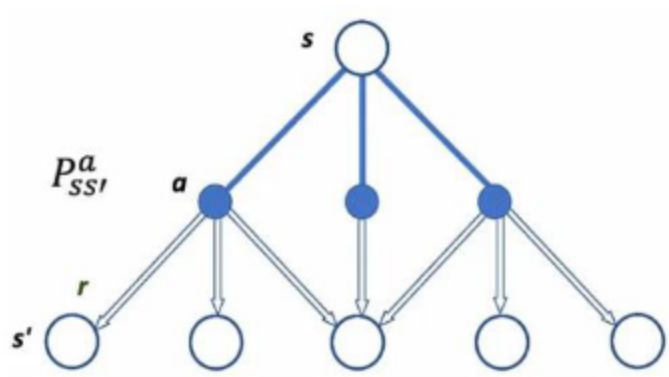
4. Write the Bellman update rule for Iterative Policy Evaluation and explain each term in it.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

Bellman Estimation equation

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\left[r + \gamma v_k(s')\right]$$

Bellman update equation for policy evaluation



(a)

Backup diagram for policy evaluation

These two are the natural backup operations for computing vπ and v∗.

where:

π(a|s) is the probability of taking action a in state s.

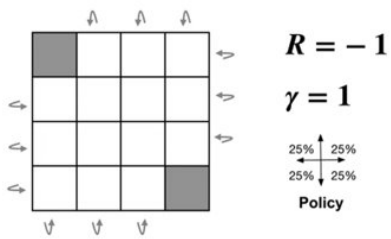
p(s',r|s, a) is the probability of moving to the next state s' and getting reward r when starting in state s and taking action a.

r is the reward received after taking this action.

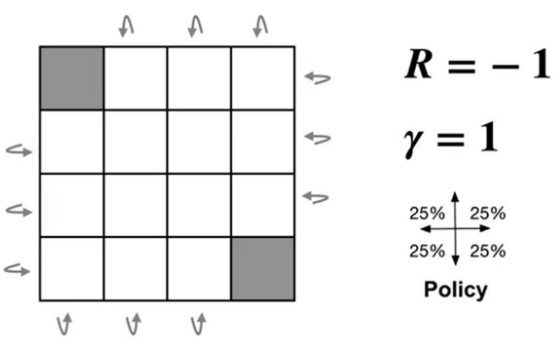
γ is the discount factor.

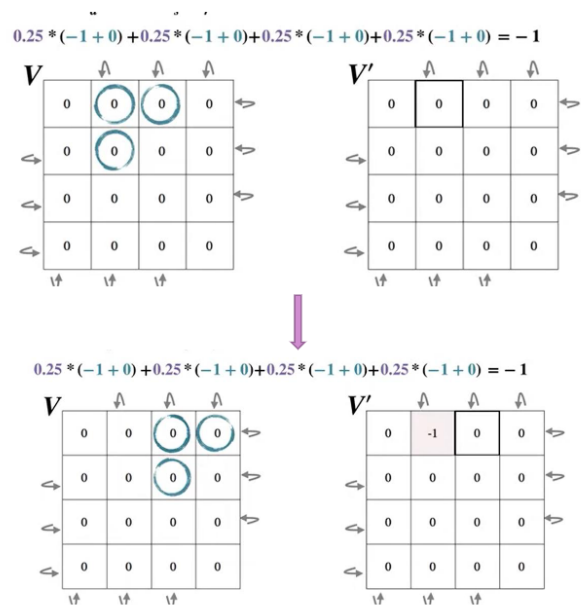
v(s') is the value of the next state.

5. Explain and solve a given Grid World problem using Iterative Policy Evaluation. (3 iterations)



Grid-world: It is a simple two-dimensional world consisting of a grid of cells, where each cell represents a state. In this problem, an agent starts in a specific cell and must navigate to a goal cell. The agent can take various actions in each state, such as moving up, down, left, or right. The goal of the agent is to find the optimal path from the starting cell to the goal cell while minimizing the total cost of the path. The cost of each action or movement in the grid world can be uniform or vary depending on the specific problem.





$$v_{k+1}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} p(s',r|s,a)\left[r + \gamma v_k(s')\right]$$

where:

π(a|s) is the probability of taking action a in state s.

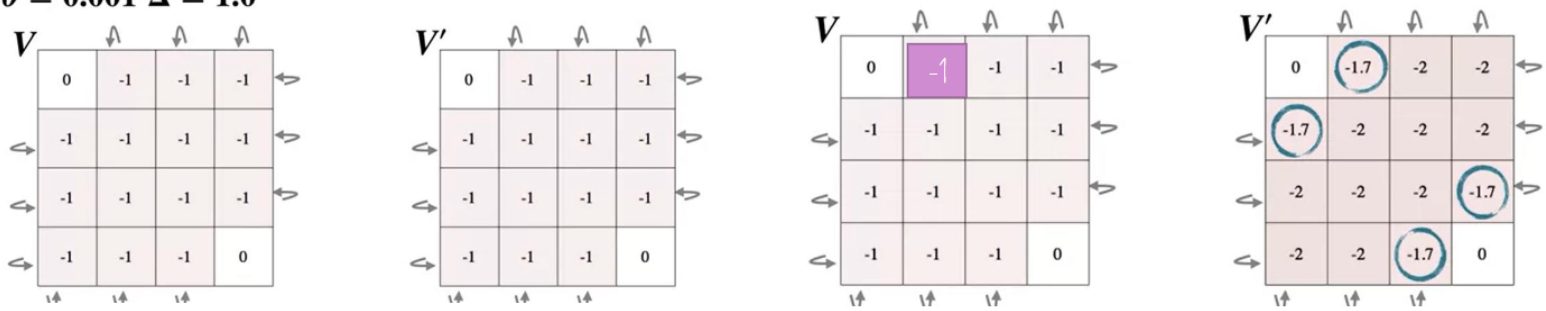
p(s',r|s, a) is the probability of moving to the next state s' and getting reward r when starting in state s and taking action a.

r is the reward received after taking this action.

γ is the discount factor.

v(s') is the value of the next state.

$\theta = 0.001 \ \Delta = 1.0$



Second iteration (k=2)

$V_{2(S_{0,1})} = 0.25*[-1+0] + 0.25*[-1+(-1)] + 0.25*[-1+(-1)] + 0.25*[-1+(-1)]$

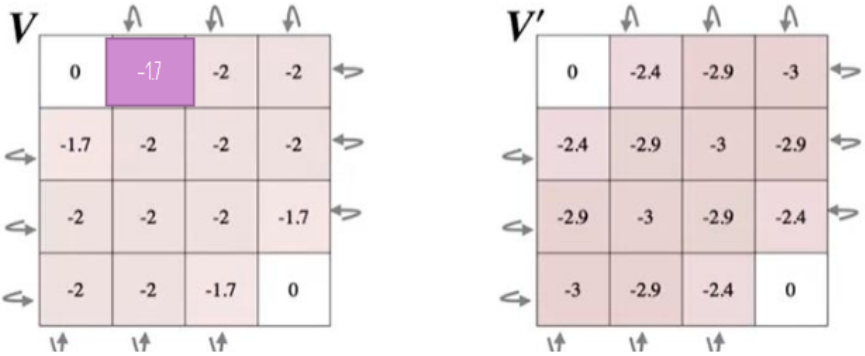
$= -0.25 -0.5-0.5-0.5$

$= -1.75 \approx -1.7$

$V_{2(S_{0,2})} = 0.25*[-1+(-1)] + 0.25*[-1+(-1)] + 0.25*[-1+(-1)] + 0.25*[-1-1]$

$= -0.5-0.5-0.5-0.5$

$= -2$



Third iteration (k=3)

$V_{3\,(S_{0,1})} = 0.25*[-1+0] + 0.25*[-1+(-1.7)] + 0.25*[-1+(-2)] + 0.25*[-1+(-2)]$

$= -0.25 -0.675-0.75-0.75$

$= -2.425 \approx -2.4$

$V_{3(S_{0,2})} = 0.25*[-1+(-1.7)] + 0.25*[-1+(-2)] + 0.25*[-1+(-2)] + 0.25*[-1+(-2)]$

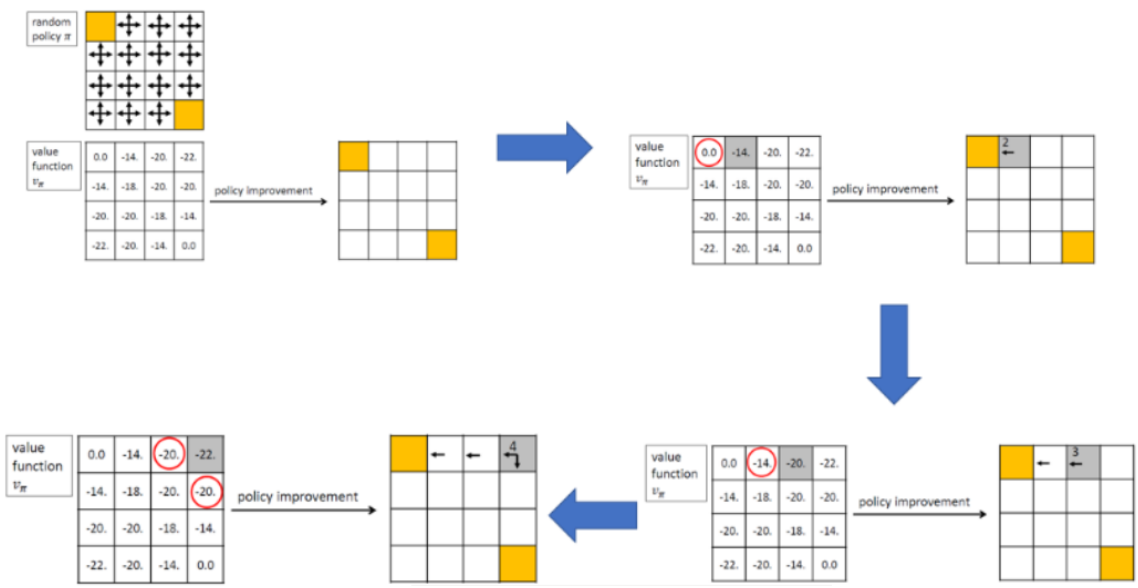
$= -0.675-0.75-0.75-0.75$

$= -2.925 \approx -2.9$

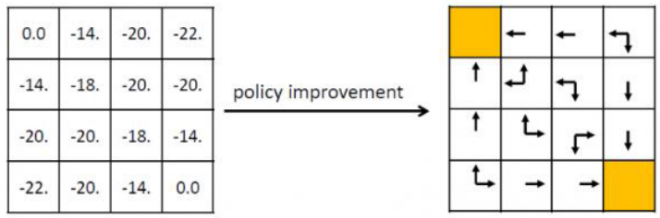
6. Explain the Policy Improvement Theorem stepwise for given V* and give the optimal policy as a result.

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

- Using v_{π}, the value function obtained for random policy π, we can improve upon π by following the path of highest value.
- We start with an arbitrary policy, and for each state one step look-ahead is done to find the action leading to the state with the highest value.
- This is done successively for each state.



Overall, after the policy improvement step using Vπ, we get the new policy π':



Looking at the new policy, it is clear that it's much better than the random policy. However, we should calculate Vπ' using the policy evaluation technique

7. Explain the concept of Policy Iteration with an example.

Policy Iteration is a method used to find the optimal policy for a given problem. It involves two main steps: policy evaluation and policy improvement.

In policy evaluation, we evaluate the performance of a given policy by computing its expected value or reward. We do this by repeatedly updating the value function of each state using the Bellman equation until convergence. In policy improvement, we improve the policy by choosing actions that maximize the value function of each state. We update the policy by selecting the action that has the highest value in each state.

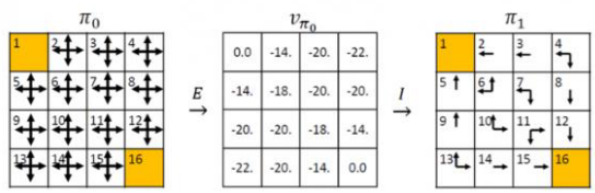
To illustrate this concept with an example of a grid world, let's consider a simple 4x4 grid world where the agent can move up, down, left, or right. The goal of the agent is to reach a goal cell in the grid world.

To apply policy iteration, we first initialize a random policy, such as selecting each action with equal probability in each state. Then, we start the iterative process by evaluating the performance of this policy. We compute the expected value of each state using the Bellman equation and update the value function until convergence.

Once we have evaluated the policy, we can improve it by selecting the action that has the highest value in each state. We update the policy by choosing the action that leads to the cell with the highest value.

Once the policy has been improved using v_π to yield a better policy π', we can then compute v_π' to improve it further to π''. Repeated iterations are done to converge approximately to the true value function for a given policy π (policy evaluation). Improving the policy as described in the policy improvement section is called policy iteration.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$



E denotes policy evaluation and I denotes policy Iteration

8. Wrrite the pseudocode for Policy Iteration.

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad a \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   $\quad$ If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

9. What are the drawbacks of Policy iteration.

One drawback to policy iteration is that each of its iterations involves policy evaluation. The policy evaluation may itself be a protracted iterative computation requiring multiple sweeps through all the states. If policy evaluation is done iteratively, then convergence exactly equal to v_π occurs only in the limit.

10. Explain the concept of Value Iteration with the help of its backup rule.

One drawback to policy iteration is that each of its iterations involves policy evaluation. The policy evaluation may itself be a protracted iterative computation requiring multiple sweeps through all the states. If policy evaluation is done iteratively, then convergence exactly equal to v_π occurs only in the limit. Value Iteration allows for stopping the iterative process before exact convergence, thus finding an approximate solution to the optimal value function in a more efficient manner.

- Let's see how this is done as a simple backup operation:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

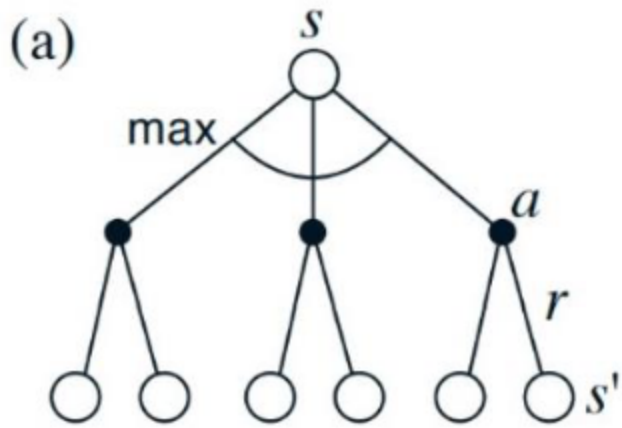
- This is identical to the bellman update in policy evaluation, with the difference being that we are taking the maximum over all actions.
- Once the updates are small enough, we can take the value function obtained as final and estimate the optimal policy corresponding to that.

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right]$$

Bellman Optimality equation

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

Bellman update equation for Value Iteration



Backup diagram for value iteration

11. Write the pseudocode for Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
$\quad \Delta \leftarrow 0$
$\quad$ For each $s \in \mathcal{S}$:
$\qquad v \leftarrow V(s)$
$\qquad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
$\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\quad \pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

12. Compare Value Iteration and Policy Iteration

Value Iteration is an iterative algorithm that directly computes the optimal value function and policy, while Policy Iteration alternates between policy evaluation and policy improvement steps. Value Iteration converges faster but requires computing the value function at each iteration, while Policy Iteration provides a clearer separation of evaluation and improvement steps and can be more efficient in large or complex MDPs.

RELATION BETWEEN VALUE ITERATION AND POLICY EVALUATION

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

Bellman Estimation equation

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_*(s')]$$

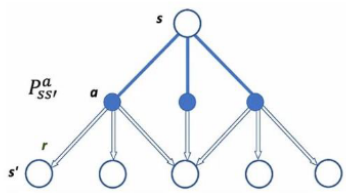
Bellman Optimality equation

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\big[r + \gamma v_k(s')\big]$$

Bellman update equation for policy evaluation

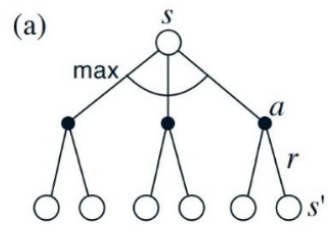
$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

Bellman update equation for Value Iteration



(a)

Backup diagram for policy evaluation



Backup diagram for value iteration

These two are the natural backup operations for computing vπ and v∗.

13. Explain the relation between policy and value iteration using backup diagrams.

RELATION BETWEEN VALUE ITERATION AND POLICY EVALUATION

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

Bellman Estimation equation

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_*(s')]$$

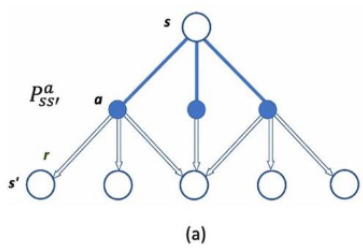
Bellman Optimality equation

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\big[r + \gamma v_k(s')\big]$$

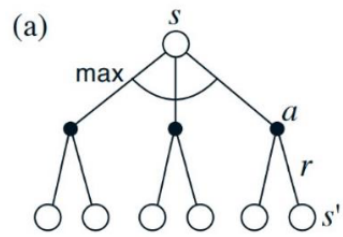
Bellman update equation for policy evaluation

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

Bellman update equation for Value Iteration



Backup diagram for policy evaluation



Backup diagram for value iteration

These two are the natural backup operations for computing vπ and v∗.

14. What is Asynchronous DP?

A major drawback to the DP methods is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set. If the state set is very large, then even a single sweep can be prohibitively expensive.

Asynchronous DP algorithms are in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set. These algorithms back up the values of states in any order whatsoever, using whatever values of other states happen to be available. The values of some states may be backed up several times before the values of others are backed up once. To converge correctly, however, an asynchronous algorithm must continue to back up the values of all the states: it can't ignore any state after some point in the computation. Asynchronous algorithms allow great flexibility in selecting states to which backup operations are applied.

Avoiding sweeps does not necessarily mean that we can get away with less computation. It just means that an algorithm does not need to get locked into any hopelessly long sweep before it can make progress in improving a policy. We can try to take advantage of this flexibility by selecting the states to which we apply backups so as to improve the algorithm's rate of progress. We can try to order the backups to let value information propagate from state to state in an efficient way. Some states may not need their values backed up as often as others. We might even try to skip backing up some states entirely if they are not relevant to optimal behavior.

Asynchronous algorithms also make it easier to intermix computation with real-time interaction. To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP. The agent's experience can be used to determine the states to which the DP algorithm applies its backups. At the same time, the latest value and policy information from the DP algorithm can guide the agent's decision-making. For example, we can apply backups to states as the agent visits them. This makes it possible to focus the DP algorithm's backups onto parts of the state set that are most relevant to the agent.

15. Which method can be used to solve the Gambler's Problem – Policy Iteration or Value Iteration? Justify.

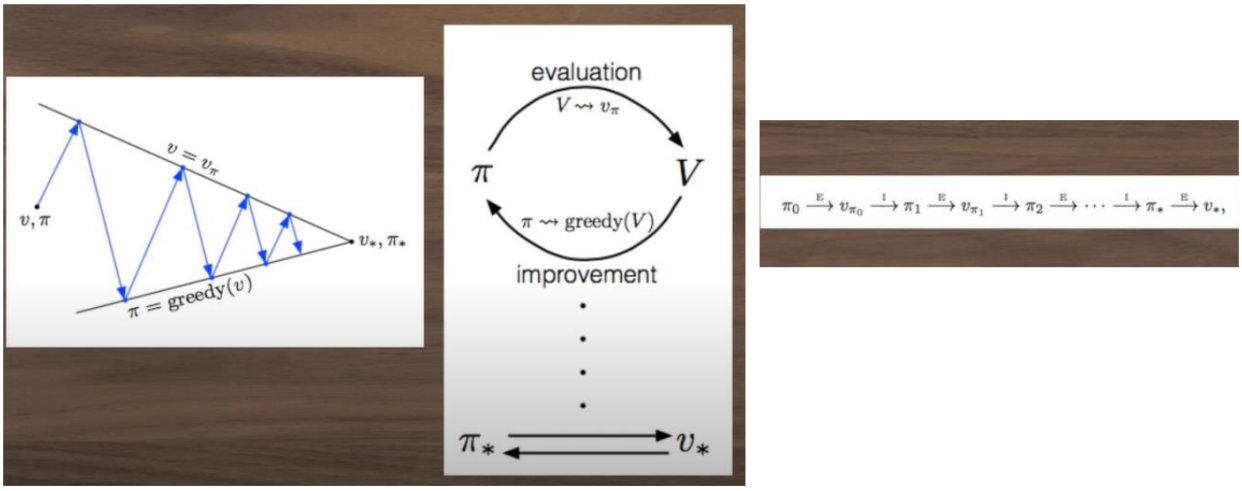
Both Policy Iteration and Value Iteration can be used to solve the Gambler's Problem. However, Value Iteration is generally preferred because it is a more efficient algorithm for finding the optimal value function, particularly in problems where the state space is large.

Value Iteration computes the optimal value function by iteratively updating the value of each state using the Bellman optimality equation until the values converge to the true optimal values. On the other hand, Policy Iteration involves iteratively improving the policy by first evaluating the value function for a given policy and then updating the policy based on the current value function. This process continues until the policy converges to the optimal policy.

16. What is GPI? Explain with proper diagram.

GPI(Generalized Policy Iteration) works as follows:

1. We randomly initialize our value function estimates of every state and start with a random policy.
2. We then evaluate the values of every state using this policy.
3. And we update the policy by making greedy action choices, with respect to the value functions (i.e., taking the action that moves you to the state with the highest value). This will continue until it converges to the optimal policy and value function.



One might also think of the interaction between the evaluation and improvement processes in GPI in terms of two constraints or goals–for example, as two lines in two-dimensional space.

Although real geometry is much more complicated than this, the diagram suggests what happens in the real case. Each process drives the value function or policy toward one of the lines representing a solution to one of the two goals. The goals interact because the two lines are not orthogonal. Driving directly towards one goal causes some movement away from the other goal. Inevitably, however, the joint process is brought closer to the overall goal of ambiguity. The arrows in the above diagram correspond to the behavior of policy iteration in that each takes the system all the way to achieving one of the two goals completely. In GPI, one could also take smaller, incomplete steps toward each goal. In other cases, the two processes together achieve the overall goal of optimality even though neither is attempting to achieve it directly.

17. Compare model-based and model-free learning. Give an example of each

18. What is the concept of bootstrapping? Is it used in DP or in MC method?

When you estimate something based on another estimation, it is called bootstrapping.

Bootstrapping is used in both dynamic programming (DP) and Monte Carlo (MC) methods in reinforcement learning. In DP methods such as value iteration and policy iteration, bootstrapping is used to update the value function of a state based on the estimated value of the successor states. This allows for the efficient calculation of the optimal value function and policy.

In MC methods such as Monte Carlo control and SARSA, bootstrapping is used to update the action-value function by using the estimated value of the next state and action instead of waiting for the final outcome of the episode. This allows for a more efficient update of the action-value function, as it only requires the knowledge of the rewards and the next state.

19. What are the advantages of Monte Carlo methods over DP?

DP Vs Monte Carlo

- Dynamic programming requires a complete knowledge of the environment or all possible transitions, whereas Monte Carlo methods work on a sampled state-action trajectory on one episode.
- DP includes only one-step transition, whereas MC goes all the way to the end of the episode to the terminal node.
- One important fact about the MC method is that the estimates for each state are independent, which means the estimate for one state does not build upon the estimate of any other state, as in the case of DP.

Advantages of MC

- The computational expense of estimating the value of a single state is independent of the number of states. This can make Monte Carlo methods particularly attractive when one requires the value of only one or a subset of states.
- One can generate many sample episodes starting from the states of interest, averaging returns from only these states ignoring all others.

20. What are MC prediction methods? Explain with example.

Monte Carlo Prediction

- We've seen that the value of a state is the expected return starting from that state and then following a particular policy.
- An easy way to estimate it based on experience, would be to average the returns we observe after visiting a state.
- As we interact with the environment more and observe more returns, the average should converge to the expected value.
- That's the idea behind all Monte Carlo methods.

Monte Carlo Prediction

- Suppose we want to estimate the value of a state $V\pi(s)$.
- Each time we visit state "s" in an episode is called a "visit to s". Clearly, we may visit "s" many times in a single episode. Let's call the first time we visit "s" in an episode the "first visit to s".
- In **first-visit MC**, we estimate the value of state "s" by averaging the returns we observe after our first visit to "s".
- In **every-visit MC**, we estimate the value of "s" by averaging the returns after all visits to "s".

21. Write the pseudocode for First-visit Monte Carlo Prediction.

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

$$\bar{V}_{\pi}(s) = \frac{1}{N} \sum_{i=1}^{N} G_{i,s}$$

- i – Episode index
- s – Index of state

22. Write the pseudocode for Every-visit Monte Carlo Prediction.

23. Calculate value of states 'A' and 'B' for given scenario (A and B are states and numbers are rewards) using Every visit MC prediction.

$$A+3 \rightarrow A+2 \rightarrow B-4 \rightarrow A+4 \rightarrow B-3 \rightarrow \text{terminate}$$

$$B-2 \rightarrow A+3 \rightarrow B-3 \rightarrow \text{terminate}$$

$$A+3 \rightarrow A+2 \rightarrow B-4 \rightarrow A+4 \rightarrow B-3 \rightarrow \text{terminate}$$

$$B-2 \rightarrow A+3 \rightarrow B-3 \rightarrow \text{terminate}$$

Every Visit MC: summing all the rewards coming after every occurrence of A (after every visit to A) for all episodes. So can have more than one summation terms per episode.

e.g.

For 1st Episode: = [3 + 2 +(-4) +4+(–3)]**+**[2+(-4)+4+(-3)]**+**[4+(-3)] = 2+(-1)+1

For 2nd Episode: = 3+(-3)=0

$$\boxed{\bar{V}_{\pi}(s) = \frac{1}{N}\sum_{i=1}^{N} G_{i,s}}$$

V(A)= (2+(-1)+1+0)/4 = ½ = 0.5

Every visit
V(A) = 1/4(2 + -1 + 1 + 0) = 1/2
V(B) = 1/4(-3 + -3 + -2 + -3) = -11/4

24. Identify the RL elements for the Blackjack Problem along with the rules of the game.

Blackjack Rules:

- 2 Players (Dealer vs. Player)
- Both players receive 2 cards at the beginning of the game
- One of the dealer's cards is face down at the beginning of the game
- Points: **Ace** = 1, **J, Q, K** = 10, All other cards equal their shown value
- The player closest to the goal wins
- If a player exceeds 21, they lose
- If both players exceed 21, it is a draw
- Sampling with replacement

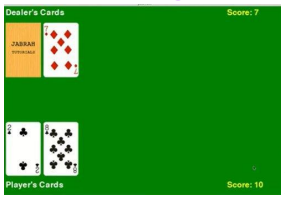
State Representation: Player's Current Total, Dealers' showing card

Action: Hit (Request for the additional card), Stick (End your turn)

Reward

- **Win** = 1, **Lose** = -1, **Draw** = 0
- All rewards are given at the end of episode

25. What are the factors to be considered by Blackjack player while taking decision of Hit or Stick? For given instances of game, identify the action chosen by the player.

A) 

B)

A) Hit

B) Stick

In Blackjack, players must decide whether to hit (request another card) or stick (stand with their current hand) based on the cards they have been dealt and the dealer's up-card. The following factors should be considered by a Blackjack player when making this decision:

The player's current hand value: The total value of the player's current hand should be considered when deciding whether to hit or stick. *If the value of the hand is low (less than 12), the player should generally hit, as there is a high chance of improving the hand. If the value of the hand is high (17 or above), the player should generally stick, as there is a high chance of busting.*

The dealer's up-card: The dealer's up-card is an important factor to consider when deciding whether to hit or stick. *If the dealer has a low up-card (2-6), the player should generally stick on a hand value of 12 or above, as there is a high chance of the dealer busting. If the dealer has a high up-card (7-Ace), the player should generally hit until they have a hand value of 17 or higher, as the dealer is less likely to bust.*

The composition of the player's hand: The composition of the player's hand is also important when making a decision to hit or stick. *If the player has a soft hand (an Ace and a low-value card), they should generally hit until they have a hand value of 18 or higher. If the player has a hard hand (no Ace or an Ace that must be counted as 1), they should generally stick on a hand value of 12 or higher.*

The number of decks in play: The number of decks in play can also impact the decision to hit or stick. In games with fewer decks, the player may be more likely to hit, as there is a greater chance of receiving a valuable card. In games with more decks, the player may be more likely to stick, as the deck is more diluted and the chances of busting are higher.

26. Explain the term usable ace with an example

In the game of Blackjack, an "ace" card can have a value of 1 or 11, whichever is more advantageous for the player.

A "usable ace" is an ace card that can be counted as 11 without causing the hand to bust. This means that the hand can still be played even if the player draws a card that would normally cause the hand to bust. For example, if a player has an ace and a 7 (totaling 18), they have a usable ace. If the player then draws a card with a value of 10 (e.g., a Jack or a Queen), their hand value becomes 28, but the ace can be counted as 1, so the hand value is reduced to 18, and the player can continue to play.

The term "usable ace" is important because it gives the player more flexibility in their play and can increase their chances of winning the game. However, it is important to note that not all aces are usable aces, and the player must use their judgment to decide whether to count an ace as 1 or 11 based on the current state of the game.

27. What are the assumptions made to obtain this guarantee of convergence for the Monte Carlo method?

2 Assumptions made to obtain this guarantee of convergence for the Monte Carlo method:

1. Episodes having exploring starts
2. Policy evaluation could be done with an infinite number of episodes

For policy evaluation to work, we need to make sure that we visit all actions in every state.
One way to go about this is to specify that every episode will start in some state-action pair and that every pair has a probability > 0 to be selected as the start pair.
This guarantees that all state-action pairs will be visited an infinite number of times in the limit of an infinite number of episodes. We call this the assumption of exploring starts.

28. Write pseudocode for Monte Carlo with Exploring Starts.

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$\pi(s) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list

Repeat forever:
Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
Generate an episode starting from S_0, A_0, following π
For each pair s, a appearing in the episode:
$G \leftarrow$ return following the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
For each s in the episode:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

29. Explain on-policy and off-policy control in Monte Carlo

On- Policy Monte Carlo Control

- In on-policy MC control methods the policy is generally ***soft*** meaning that the probability for taking action ***a*** in state ***s*** is >0 for all ***a*** in ***A(s)*** and ***s*** in ***S.***
- But over time the policy shifts gradually closer and closer to a deterministic policy.
- One way to define such a policy is using ***epsilon-greedy policies,*** which means that most of the time the action with the highest estimated value is selected, and with a probability of *epsilon* a random action is selected.
- By behaving this way a constant exploration is guaranteed.

- That is, the policy is designed in a way that all nongreedy action are given the minimal probability of selection which is:

$$\frac{\epsilon}{|A(s)|}$$

- And the remaining bulk of the probability is given to the greedy action:

$$1 - \epsilon + \frac{\epsilon}{|A(s)|}$$

- The epsilon-greedy policies are examples of epsilon-soft policies and defined as policies for which

$$\pi(a|s) \geq \frac{\epsilon}{|A(s)|}$$

- for all states and actions, for some *epsilon* > 0. Among epsilon-soft policies, epsilon-greedy policies are in some sense those that are closest to greedy.

Off-Policy Monte Carlo Control

- Off-policy Monte Carlo control methods follow the behavior policy while learning about and improving the target policy.
- These techniques requires that the behavior policy has a nonzero probability of selecting all actions that might be selected by the target policy (coverage).
- To explore all possibilities, we require that the behavior policy be soft (i.e., that it select all actions in all states with nonzero probability).

30. Explain the concept of Importance sampling with example

Importance Sampling

- Consider the 2 vectors below:

A1	0.2	A1	0.7
A2	0.6	A2	0.1
A3	0.1	A3	0.1
A4	0.1	A4	0.1

π (left) and b (right)

Looking at the target policy π, we see that we have 20% chance of taking action A1, where the behavior policy b have 70% chance of taking A1. These kind of different probabilities for actions is what we need to weight the actions differently in π (left) and b (right)

- So let's see how importance sampling is used to align the probabilities of 2 policies, so that we can update π even though we're behaving using b.

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

- Looking at the formula for the importance-sampling ratio, it is the product of taking a certain action given a certain state from the target policy compared to the behavior policy.

31. How to calculate the Importance sampling ratio?

- Given a starting state St , the probability of the subsequent state–action trajectory, At , St+1, At+1, . . . , ST , occurring under any policy π is

$$\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),$$

- Where p is the state-transition probability function. Thus, the relative probability of the trajectory under the target and behavior policies (the importance-sampling ratio) is

$$\rho_t^T = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}.$$

- Note that although the trajectory probabilities depend on the MDP's transition probabilities, which are generally unknown, all the transition probabilities cancel and drop out.
- The importance sampling ratio ends up depending only on the two policies and not at all on the MDP.

32. What are the types of importance sampling? What is the formula to calculate the state value function in each of the categories of IS?

- Generally speaking, there are two ways to calculate the state-value with importance sampling.
- **Ordinary importance sampling,** where the ration is multiplied by the Return and averaged by *T(s)* (the set of all time steps in which state s is visited):

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

- Or **weighted importance sampling**, which is a weighted average:

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

let T(t) denote the first time of termination following time t, and Gt denote the return after t up through T(t). Then {Gt}t∈T(s) are the returns that pertain to state s, and {ρ T(t) t }t∈T(s) are the corresponding importance-sampling ratios

33. What are the equations for state value estimate using Ordinary IS and Weighted IS? For given data of return at time steps and probability distribution for target and behavior policies (consider only three time steps), calculate state value i.e. V(s) using Ordinary IS and Weighted IS .

G1= 2	π 1= 1/3	μ1=4/3
G2= 3	π 2= 1/3	μ2=1/3
G3= 4	π 3= 2/3	μ3=2/3

- **Ordinary importance sampling**, where the ration is multiplied by the Return and averaged by *T(s)* (the set of all time steps in which state s is visited):

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

- Or **weighted importance sampling**, which is a weighted average:

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

let T(t) denote the first time of termination following time t, and Gt denote the return after t up through T(t). Then {Gt}t∈T(s) are the returns that pertain to state s, and {ρ T(t) t }t∈T(s) are the corresponding importance-sampling ratios

$\pi(A_k|S_k)$ – Taking certain action given a certain state under policy π.

$\mu(A_k|S_k)$ – —— " —— " —— μ.

$$\rho_t^T = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \text{ — IS ratio.}$$

$G_t \Rightarrow$ Return at time step t.

Let us consider 3 time steps.

e.g.

$G_{t=1} = 2$

$G_{t=2} = 3$

$G_{t=3} = 4$

probability distribution –

$\pi_1 = 1/3$ $\quad \mu_1 = 4/3$

$\pi_2 = 1/3$ $\quad \mu_2 = 1/3$

$\pi_3 = 2/3$ $\quad \mu_3 = 2/3$

Ordinary IS

$$V_\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|} \quad \text{ie. } \frac{1}{n}\left[\sum \frac{G_i\, \pi(x_i)}{\mu(x_i)}\right]$$

$$= \frac{1}{3}\left[2 \times \frac{1}{3}/4/3 + 3 \times \frac{1}{3}/1/3 + 4 \times 2/3/2/3\right]$$

$$= \frac{1}{3}\left[\cancel{2} \times \frac{1}{\cancel{3}} \times \frac{\cancel{3}}{\cancel{4}_2} + 3 \times 1 + 4 \times 1\right]$$

$$= \frac{1}{3}\left[\frac{1}{2} + 3 + 4\right]$$

$$= \frac{1}{\cancel{3}} \times \frac{\cancel{15}\,5}{2}$$

$$= 5/2 \quad \text{ie. } 2\tfrac{1}{2}$$

Weighted IS

$$V_\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)}} \quad \text{ie. } \frac{\sum G_i\, \pi(x_i)/\mu(x_i)}{\sum \frac{\pi(x_i)}{\mu(x_i)}}$$

$$= \frac{\left[2 \times 1/3/4/3 + 3 \times 1/3/1/3 + 4 \times 2/3/2/3\right]}{1/3/4/3 + 1/3/1/3 + 2/3/2/3}$$

$$= \frac{\frac{\cancel{2}}{\cancel{4}_2} + 3 + 4}{\frac{1}{4} + 1 + 1}$$

$$= \frac{15/2}{9/4}$$

$$= \frac{\cancel{15}\,5}{\cancel{2}} \times \frac{\cancel{4}\,2}{\cancel{9}\,3}$$

$$= 10/3 = 3\tfrac{1}{3}$$

34. Explain the Incremental Implementation of Monte Carlo

Incremental Implementation

- Monte Carlo prediction methods can be implemented incrementally, on an episode-by-episode basis using techniques we learned for bandit problems where we averaged rewards, but in Monte Carlo methods we average returns.
- In all other respects exactly the same methods can be used for on-policy Monte Carlo methods.
- For off-policy Monte Carlo methods, we need to separately consider those that use ordinary importance sampling and those that use weighted importance sampling.

Incremental Implementation

- In ordinary importance sampling, the returns are scaled by the importance sampling ratio $\rho_t^{T(t)}$, then simply averaged.
- For these methods we can again use the incremental methods we learned earlier but using the scaled returns in place of the rewards
- This leaves the case of off-policy methods using weighted importance sampling.
- Here we have to form a weighted average of the returns, and a slightly different incremental algorithm is required.

- Suppose we have a sequence of returns G1, G2, . . . , Gn−1, all starting in the same state and each with a corresponding random weight Wi (e.g., Wi = $\rho_t^{T(t)}$).

- We wish to form the estimate $$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \qquad n \geq 2,$$

- and keep it up-to-date as we obtain a single additional return Gn. In addition to keeping track of Vn, we must maintain for each state the cumulative sum Cn of the weights given to the first n returns. The update rule for Vn is

$$V_{n+1} = V_n + \frac{W_n}{C_n}\Big[G_n - V_n\Big], \qquad n \geq 1, \quad \text{and} \quad C_{n+1} = C_n + W_{n+1},$$

where C0 = 0 (and V1 is arbitrary and thus need not be specified).

35. Derive the Incremental update rule of Monte Carlo

By definition of V_{n+1}, we have:

$$V_{n+1} = \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} \tag{1}$$

Then, taking the nth term out of the sum in the numerator, we have:

$$V_{n+1} = \frac{W_n G_n + \sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n} W_k} \tag{2}$$

Then, from the definition of V_n, $V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}$, we have:

$$\sum_{k=1}^{n-1} W_k G_k = V_n * \sum_{k=1}^{n-1} W_k \tag{3}$$

Then, substituting (3) in the numerator of (2), we get:

$$V_{n+1} = \frac{W_n G_n + V_n * \sum_{k=1}^{n-1} W_k}{\sum_{k=1}^{n} W_k} \tag{4}$$

Then, adding and subtracting $V_n W_n$ in the numerator of (4), we obtain:

$$V_{n+1} = \frac{W_n G_n + V_n * \sum_{k=1}^{n-1} W_k + V_n W_n - V_n W_n}{\sum_{k=1}^{n} W_k} \tag{5}$$

We factor V_n in the numerator of (5):

$$V_{n+1} = \frac{W_n G_n + V_n (W_n + \sum_{k=1}^{n-1} W_k) - V_n W_n}{\sum_{k=1}^{n} W_k} \tag{6}$$

We simplify, taking into account that the denominator $\sum_{k=1}^{n} W_k = W_n + \sum_{k=1}^{n-1} W_k$, we get:

$$V_{n+1} = \frac{W_n G_n + V_n (W_n + \sum_{k=1}^{n-1} W_k) - V_n W_n}{\sum_{k=1}^{n} W_k} \longrightarrow V_{n+1} = V_n + \frac{W_n G_n - W_n V_n}{\sum_{k=1}^{n} W_k} \tag{7}$$

Further rearrangements of the terms give us:

$$V_{n+1} = V_n + \frac{W_n}{\sum_{k=1}^{n} W_k} [G_n - V_n] \tag{8}$$

Finally, by definition of C_n as the cumulative sum of the weights up to time n, we get the desired incremental update equation:

$$V_{n+1} = V_n + \frac{W_n}{C_n} [G_n - V_n] \tag{9}$$

36. Write pseudocode for off-policy MC control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$C(s,a) \leftarrow 0$

$\pi(s) \leftarrow$ a deterministic policy that is greedy with respect to Q

Repeat forever:

Generate an episode using any soft policy μ:

$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$

$G \leftarrow 0$

$W \leftarrow 1$

For $t = T-1, T-2, \ldots$ downto 0:

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

$W \leftarrow W \frac{1}{\mu(A_t|S_t)}$

If $W = 0$ then ExitForLoop