SIT796 Reinforcement Learning

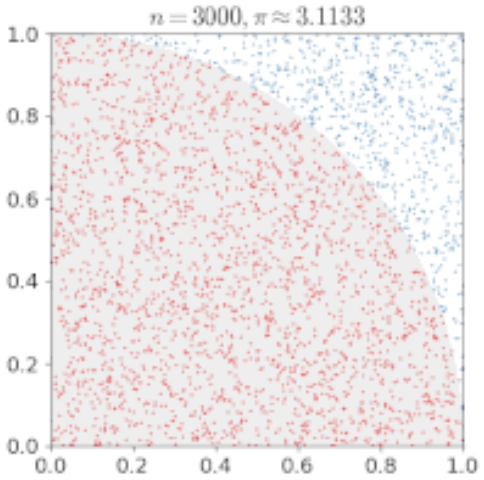
Monte Carlo Methods

DEAKIN UNIVERSITY

Presented by:
Dr. Thommen George Karimpanal
School of Information Technology

Monte Carlo Methods

Monte-carlo sampling:



The term Monte Carlo is used for any algorithm that is significantly random in its operation
Calculate π by randomly placing a dot and counting frequency of dots inside or outside the circle.

Area of circular portion: $A_C = \pi r^2/4$

Area of square: $A_s = r^2$

$$\frac{A_C}{A_S} = \pi/4$$

$$\pi = 4\frac{A_C}{A_S}$$

Monte Carlo Methods

Disadvantage of Dynamic Programming: requires a full and complete model of the MDP
Monte Carlo (MC) methods only require *experience*.

In RL, MC methods simply need to sample: states, actions and rewards from the environment

Episode: one attempt at solving the task (episodes end based on terminal condition)

MC methods are incremental in an episode-by-episode sense.

In this sense they are similar to bandits – except stretched out over a number of steps (each acting like a separate bandit problem).

Monte Carlo (Prediction)

MC can be used to learn the state value function for a given policy – as we have seen this is referred to as *prediction*

- Recall – The value of a state is the *expected cumulative future discounted reward* starting at that state (expected return).
- The simplest approach to learn this from experience is to average the observed returns after visiting the state.
- This is the notion that underlies MC.
- The following example considers the case where we only calculate for the first-visit.

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

Forward: $\{s_0, a_0, r_0\}, \{s_1, a_1, r_1\}, \{s_T, a_T, r_T\}$

G=0

G= r_0 G= $r_0 + \gamma r_1$ G= $r_0 + \gamma r_1 + \gamma^2 r_T$

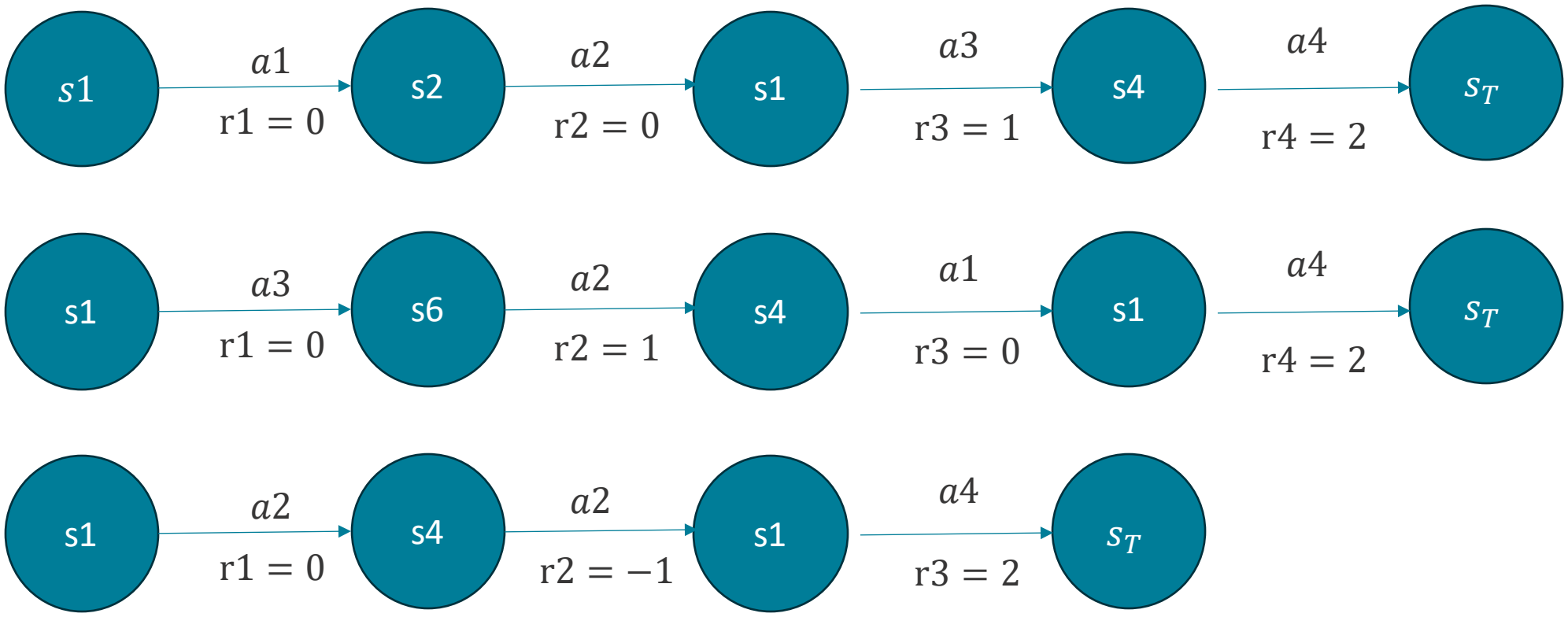
Backward: $\{s_T, a_T, r_T\}, \{s_1, a_1, r_1\}, \{s_0, a_0, r_0\}$

G=0

G= $\gamma * 0 + r_T$ G= $\gamma * r_T + r_1$ G= $\gamma^2 r_T + \gamma r_1 + r_0$

First Visit Monte Carlo (example)

3 episodes, $\gamma = 1$

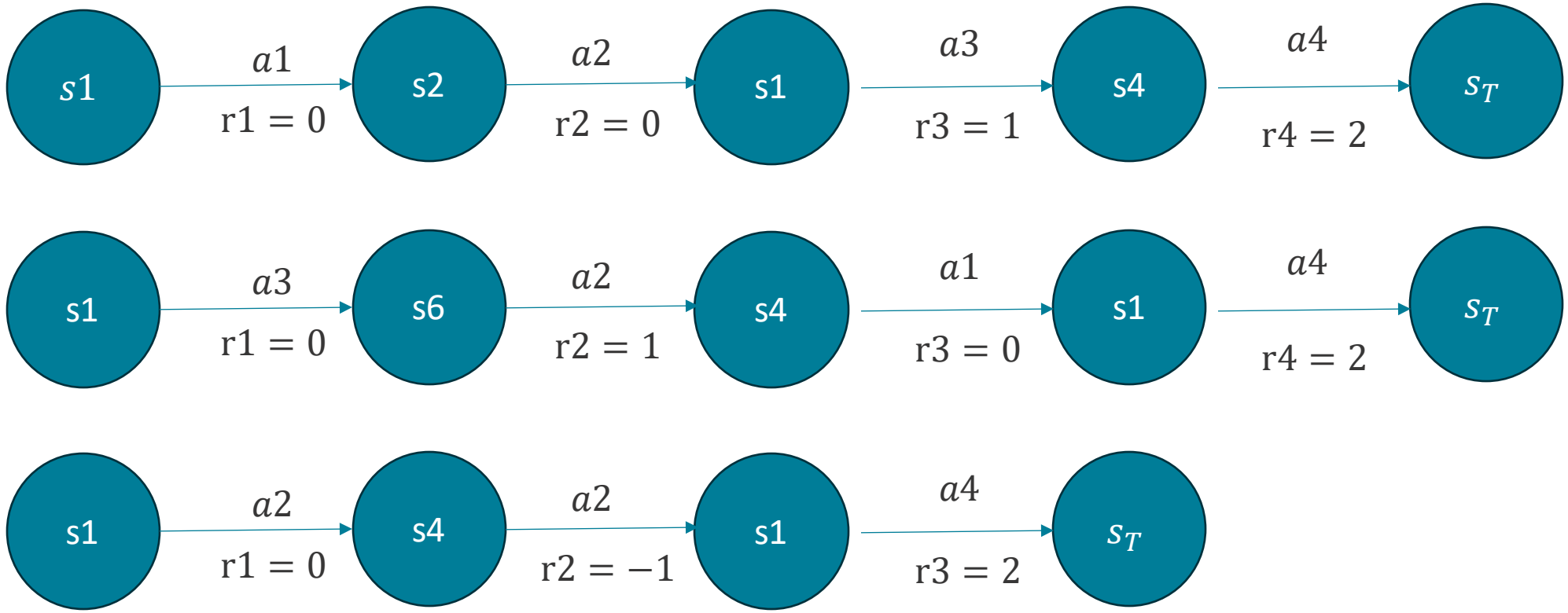


V(s1)=(3+3+1)/3=2.33

V(s4)=(2+2+1)/3=1.66

Every Visit Monte Carlo (example)

3 episodes, $\gamma = 1$

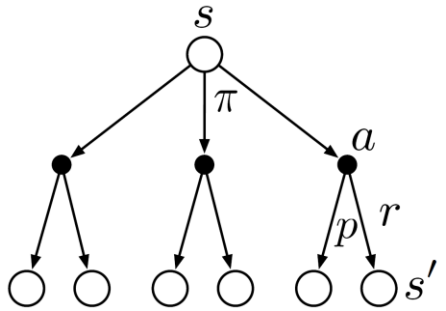


$V(\text{s1}) = [\ (0+0+1+2)+(1+2).\ +.\ (0+1+0+2)+(+2).\ +.\ (0\text{-}1+2)+(2).\ ]/6 = 2.33$

Monte Carlo (Prediction)

If we compare a MC backup diagram against that of DP, below:

- DP updates a single $V(s)$ for all actions based on the policy being followed.
- Whereas MC updates the $V(s)$ based only on the path followed by the policy.
  - Does not build upon other states estimates – no bootstrapping.



Importantly the computational cost is independent of how many states there are.

- Particularly useful if we only want to calculate a subset of states.
- Learn only from states that occur

Example – Blackjack

A card game where you play against the dealer

- Aim is to get a higher score than the dealer without going bust (> 21).
- You are both given two cards (one of the dealer's cards is initially hidden).
- if you are dealt an ace and ten (21) and the dealer doesn't have an ace or a ten then you win immediately
- You can keep drawing more cards (called a hit) until you have as high as you want to risk.
  - If you draw too many and go over 21 you have lost
- Once you have drawn all the cards you desire you finish (called stick)
- The dealer then reveals the hidden card
- The dealer must keep drawing a card until they are ≥ 17.
  - If the dealer goes over 21 then you automatically win
- Once the dealer finishes drawing cards, if they arent bust
  - you win if your score is higher
  - You draw (keep your money) if your sums are the same
  - Lose if your sum is less
- Note an Ace (can be 1 or 11) allows us to take a risk (when 11) even when we have a good score

The difficulty is knowing when to draw based on what you have and the one card you can see the dealer has.

https://www.blackjack.org/wp-content/uploads/2018/12/Blackjack-values-705x396.png

https://thumbor.forbes.com/thumbor/960x0/https%3A%2F%2Fspecials-images.forbesimg.com%2Fdam%2Fimageserve%2F1051931270%2F960x0.jpg%3Ffit%3Dscale

Example – Blackjack with MC Prediction

Let's consider a policy that we want to test

- The policy sticks if the player has a sum of 20 or 21
- Otherwise it always hits

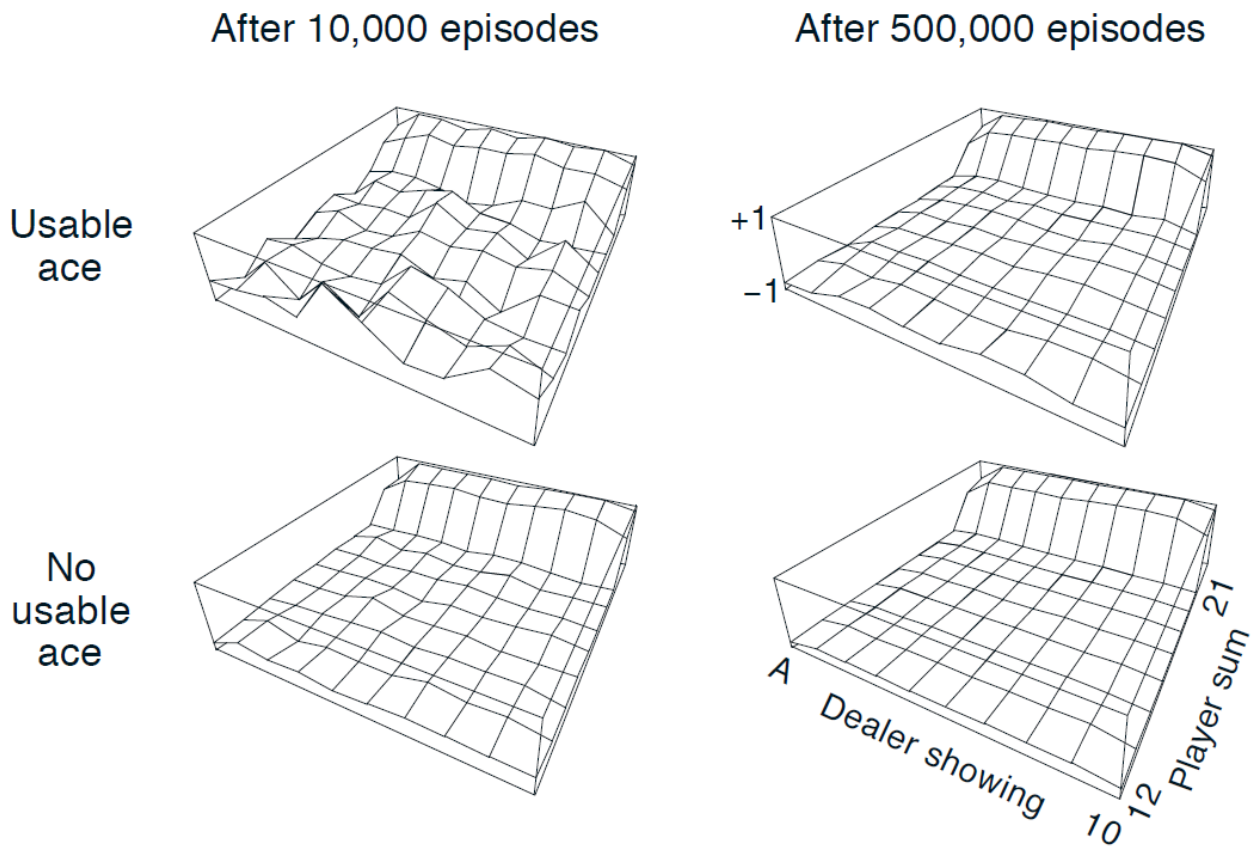
To find the state value of this policy using MC prediction we can simulate many games using the policy

- average the returns observed.

Why does the estimated value function jump up for the last two rows in the rear?

Why does it drop off for the whole last row on the left?

Why are the frontmost values higher in the upper diagrams than in the lower?



Sutton and Barto (2018)

SIT796 Reinforcement Learning

Monte Carlo Control

DEAKIN UNIVERSITY

Presented by:
Dr. Thommen George Karimpanal
School of Information Technology

Monte Carlo with Action Values

In DP we calculated state values because we can simply select the action that leads to the best next state.

- This is possible because we have a model of the environment – hence we can track what states come next and their value.
- However if we don't have a model we can not do this.
- If we can estimate the value of actions taken from a particular state (rather than just the value of the state), we can just choose the best actions based on the Q value.
- Often referred to as state-action values and denoted $Q_\pi(s, a)$
- MC, with enough experience can estimate Q^*

In reality we do this by visiting state-action pairs multiple times to estimate the value.

- Therefore, MC finds the average of the returns from that state-action pair.
- Some state-action pairs may never be visited. Eg if π never uses an action we will never learn a value for it.
  This fits with the general problem we discussed with multi-armed bandits – maintaining exploration
- *Exploring starts* – starting at a random state-action pair, where there is a probability of starting in any state-action pair
- Use a *stochastic policy* where all state-actions have a probability of being selected.

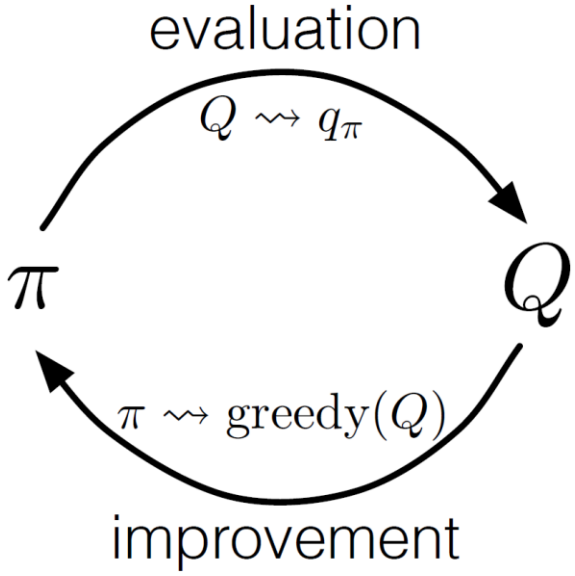
Monte Carlo (Control)

Recall, Control is a search for a policy

- whereas prediction was aiming to find the value – given a policy.

Recall the GPI approach in DP

- Recall, GPI maintains both a policy and an approximate value function.
  - The value function is updated to approach the value function of the current policy
  - The policy is improved based on the current value function
- Hence these work against each other to some degree



In MC we can follow a similar approach

- Again we can alternate between policy evaluation and policy iteration.

$$\pi_0 \overset{E}{\rightarrow} q_{\pi_0} \overset{I}{\rightarrow} \pi_1 \overset{E}{\rightarrow} q_{\pi_1} \overset{I}{\rightarrow} \pi_2 \overset{E}{\rightarrow} \cdots \overset{I}{\rightarrow} \pi_* \overset{E}{\rightarrow} q_*$$

- Except now we apply the evaluation based on the the complete trajectory.
- Policy iteration is performed by selecting the greedy policy with respect to the current value function
- This is also applied on the state-action values instead of state values and hence do not require a model

$$\pi(s) = \arg\max_a q(s, a)$$

Monte Carlo (Control): Exploring Starts (ES)

If we assume exploring starts and an infinite number of iterations

- Then MC is guaranteed to converge using the following algorithm

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

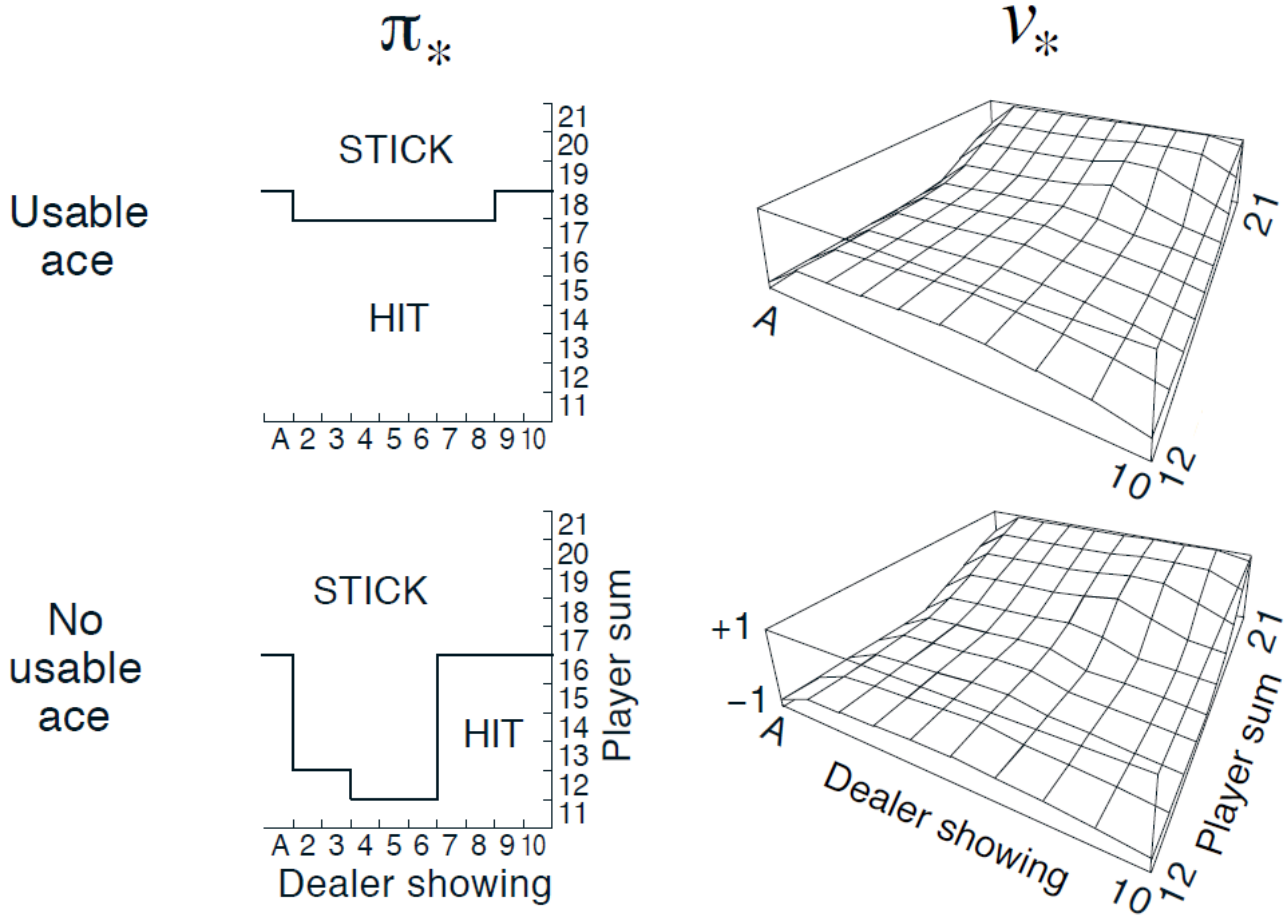
Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Example – Solve Blackjack with MC Control

Revisiting the Blackjack problem – it is now easy to fully solve using MC Control

- Can easily arrange for exploring starts.
- Cycle through all the possible start positions and simulate what happens.
- Give all state-action values an initial value – say 0.
- Apply the algorithm

The final strategy (left) is almost identical to the expert strategy of Thorp(1966)



Sutton and Barto (2018)

Monte Carlo (Control): On-Policy without ES

Exploring starts is a form of On-Policy Control

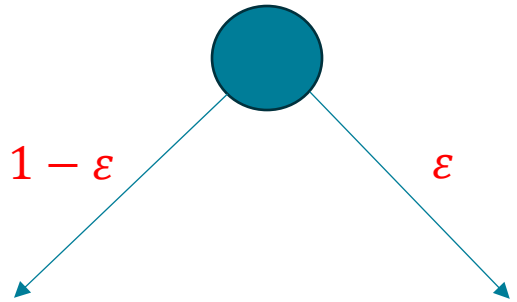
- ***On-Policy*** – evaluates and improves the policy being followed when making decisions.

However, ES is fundamentally an unlikely assumption for most tasks.

- To avoid ES we need a way of visiting every state-action pair
- Can be done using a stochastic policy. That is $\pi(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
- Then overtime shift this stochastic policy more towards a deterministic policy
- In our discussion of multi-armed bandits we discussed some ways of doing this:
  - $\varepsilon - greedy$, Upper-Confidence-Bound, soft-max.
  - There are many others eg *Thomson Sampling,* we have not discussed.

If we assume $\varepsilon - greedy$ we say this is a stochastic policy because:

- Non-greedy actions will have ${}^{\varepsilon}/_{|\mathcal{A}(s)|}$ probability of being selected
- The greedy action will have $1 - \varepsilon + {}^{\varepsilon}/_{|\mathcal{A}(s)|}$ probability of being selected.
- Therefore, $\pi(a|s) \geq {}^{\varepsilon}/_{|\mathcal{A}(s)|}$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, which aligns with the above condition – providing $\varepsilon > 0$
- Any of the above $\varepsilon - soft$ policies also provide convergence guarantees as stochastic policies.



But with probab. ${}^{\varepsilon}/_{|\mathcal{A}(s)|}$, this random action can be the best action.

Thus, total probability of best action:

$$1 - \varepsilon + {}^{\varepsilon}/_{|\mathcal{A}(s)|}$$

Monte Carlo (Control): On-Policy without ES

MC On-Policy Control is still GPI

- Our $\varepsilon - soft$ policy should move towards a greedy policy but as we can not actually achieve infinite trials we can not actually reach a fully greedy policy.
- Hence our $\varepsilon - soft$ policy should never actually reach greedy

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

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

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

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

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

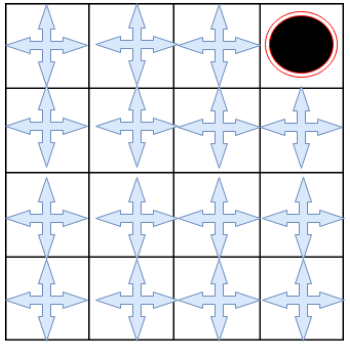
For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

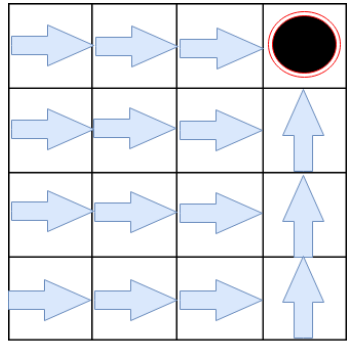
On/Off-Policy Learning

On-Policy evaluates and improves the policy being followed.

- The issue is that the agent must perform non-optimal behaviour to learn about all state-actions
- This means the state-action values being learnt is based on the current policy (which may not be optimal)
  - Hence, may learn significantly inferior values.
  - May think a state value is not-optimal (when it is) because the later actions were exploratory.



Behavior Policy



Target Policy

Off-Policy instead evaluates and improves the optimal (target) policy instead of the one followed. Allows the agent to explore without damaging the state-action values being learnt.

To do this you effectively maintain two policies

The current optimal policy being learnt – called the ***target policy***

A ***behaviour policy*** that generates the decisions and can be more exploratory

In this way, the state-action values learn from data "off" the target policy, hence the approach is called ***off-policy learning***

In this unit we will consider both on-policy and off-policy approaches to different algorithms

On-policy is generally simpler and is considered first.

You can think of On-policy as simply having the same behaviour and target policies – hence is a special case of off-policy

Generally, on-policy learns faster, but off-policy is often more powerful

Off-Policy Monte Carlo Prediction

Let's suppose we have collected months of data about how your household uses its solar power cell.

- This data represents a policy – the behaviour followed by your house (Behaviour Policy)

Let's say we want to learn the optimal policy so we can advise you where to save power

- How do we do this when we only have data for your actual behaviour and not the optimal behaviour?

Hence, we want to learn a policy, v_π or q_π, but only have a policy b, where $b \neq \pi$

This can be done using ***off-policy learning***.

- However, it does require the *assumption of coverage* to be met:
- E.g. we require every action to be taken under π must also be taken at least sometimes in b.
  - Hence, $\pi(a|s) > 0 \Rightarrow b(a|s) > 0$
- Therefore, our behaviour policy, b should be at least to some degree stochastic.

If our problem meets the assumption of coverage then we can learn the target policy π from our observations of b using importance sampling.

Importance Sampling

Importance Sampling allows us to estimate expected values under one distribution given samples from a separate distribution.

- In off-policy learning we weight the returns observed based on the relative probability of their future trajectories occurring under both the target and behaviour policies.
- This is called the ***importance-sampling ratio***
- Given the start state S_t, we want to calculate the probability of the future path, $A_t, S_{t+1}, A_{t+1}, \dots, S_T$,occurring under any policy π using

$$Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T | S_t, A_{t:T-1} \sim \pi\}$$

$$= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1})$$

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

Where p is the state transition probability

Therefore, the relative probability or ***importance-sampling ratio*** is calculated as:

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

Note: while the transition probabilities of the MDP are unknown, because they are the same for the numerator and denominator they cancel

Ordinary Importance Sampling

Recall, we aim to find the state or state-action values (expected returns) of the target policy.

- However, the returns G_t generated by the behaviour policy are incorrect $\mathbb{E}[G_t|S_t = s] = v_b(s)$ and so can not be directly used to calculate v_π.
- Instead, we use the ***importance-sampling ratio*** to transform the returns to the correct expected value.

$$\mathbb{E}[\rho_{t:T-1}\, G_t|S_t = s] = v_\pi(s)$$

- E.g. simply multiply our observed return by the importance-sampling ratio.
- To calculate $v_\pi(s)$ we use:

Importance sample ratio for first termination following time t

The return after episode completes

$$v_\pi(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}\, G_t}{|\mathcal{T}(s)|}$$

Sum each first visit of s

The number of episodes a state s is visited at least once

- This equation has effectively joined each episode end to end (time steps continue across episode boundaries).
- This simple average of samplings is called an ***ordinary importance sampling***

Weighted Importance Sampling

A powerful alternative is a ***weighted importance sampling*** using a weighted average.

$$v_\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} \, G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

- If the importance of the sample (denominator) is zero then $v_\pi(s) = 0$.
- In practice, the ***weighted importance sampling*** has lower variance and is preferred.
  - The ***ordinary importance sampling*** is easier to apply when we look at function approximation

SIT796 Reinforcement Learning

Monte Carlo: Off-policy Estimation

DEAKIN UNIVERSITY

Presented by:
Dr. Thommen George Karimpanal
School of Information Technology

Incremental Implementation

Recall, when tracking our Q-value over time in our multi-armed bandit (Topic 4), we could maintain an incremental average.

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$$

- Therefore, we only needed to store the current Q_n and the value of n

We can perform a similar approach when using ***ordinary importance sampling*** except use the scaled return in place of the reward. E.g.

$$V_{n+1} = V_n + \frac{W_i}{n}[\,G_t - V_n]$$

- Where $W_i = \rho_{t_i:T(t_i)-1}$, e.g. the weighting of a particular sample.

When using the weighted importance sampling however we need to perform a weighted average of the returns.

- Now we need to also track the cumulative sum C_n, where $C_{n+1} = C_n + W_{n+1}$, of the weights given and use this instead of n. E.g.

$$V_{n+1} = V_n + \frac{W_n}{C_n}[\,G_t - V_n]$$

- Where W_n is the weighting of the sample in the n^{th} episode and C_n is the cumulative weights up to and including the n^{th} episode.

Off-Policy Monte Carlo Prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$

Loop forever (for each episode):
  $b \leftarrow$ any policy with coverage of π
  Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Off-Policy Monte Carlo Control

Recall, On-policy Control evaluates and improves the policy being followed when making decisions.

In Off-Policy the policy used to generate *behaviour* may be different to that is being evaluated as the *target*.

- Advantage of this approach is target policy can be deterministic (greedy) while the behaviour policy can sample other actions.

Off-Policy MC Control can use one of the two methods discussed for Off-policy MC Prediction

- As we require the behaviour policy to cover all possible state-actions that might be required by the target policy, we must have a nonzero probability of selecting all actions.
  - This is required to insure the *assumption of coverage.*
  - To do this we must use an $\varepsilon - soft$ behaviour policy

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \in \mathbb{R}$ (arbitrarily)
$C(s,a) \leftarrow 0$
$\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
$b \leftarrow$ any soft policy
Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
$W \leftarrow W \frac{1}{b(A_t|S_t)}$

Readings

This lecture focused on Monte Carlo methods for prediction and control.

- Same capabilities as DP without a transition model!
- However, it needs full trajectories/episodes – TD learning will solve this issue.

For more detailed information see Sutton and Barto (2018) Reinforcement Learning: An Introduction

- *Chapter 5: Monte Carlo Methods*
- http://incompleteideas.net/book/RLbook2020.pdf