→ This is because the distribution of the behavior policy and the target policy will be different. So, to correct this, we introduce a new technique called importance sampling. This is a technique for estimating the values of one distribution when given samples from another.

Off-policy Monte Carlo method algorithm:-

Step 1:- Initialize the Q function $Q(s,a)$ with random values, set the behavior policy b to be epsilon-greedy, and target policy π to be greedy policy and initialize the cumulative weights as $C(s,a) = 0$

Step 2:- For M number of episodes:

1. Generate an episode using the behavior policy b.
2. Initialize return R to 0 and weight W to 1
3. For each step t in the episode, $t = T-1, T-2, \ldots, 0$:
    1. Compute the return as $R = R + r_{t+1}$
    2. Update the cumulative weights $C(s_t, a_t)$,
    $$C(s_t, a_t) = C(s_t, a_t) + W$$
    3. Update the Q value as
    $$Q(s_t, a_t) = Q(s_t, a_t) + \frac{W}{C(s_t, a_t)}\left(R_t - Q(s_t, a_t)\right)$$
    4. Compute the target policy $\pi(s_t) = \arg\max_a Q(s_t, a)$
    5. If $a_t \neq \pi(s_t)$ then break
    6. Update the weight as $W = W \dfrac{1}{b(a_t|s_t)}$

Step 3:- Return the target policy π.

→ In the on-policy method, we generate an episode using the policy π and we improve the same policy π iteratively to find the optimal policy.

→ But in the off-policy method, we generate an episode using a policy called the behavior policy b and we try to iteratively improve a different policy called the target policy π.

Algorithm:-

Step 1. Initialize the Q function $Q(s,a)$ with random values, set the behavior policy b to be epsilon-greedy, and also set the target policy π to be greedy policy

Step 2:- For M number of episodes:

1. Generate an episode using the behavior policy b
2. Initialize return R to 0.
3. For each step t in the episode, $t = T-1, T-2, \ldots, 0$.
   1. Compute the return as $R = R + r_{t+1}$
   2. Compute the Q value as
      $$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t))$$
   3. Compute the target policy $\pi(s_t) = \arg\max_a Q(s_t, a)$
      $$\pi(s_t) = \arg\max_a Q(s_t, a)$$

Step 3:- Return the target policy π.

Issue:- Since we are finding the target policy π from the Q function, which is computed based on the episodes generated by a different policy called the behavior policy, our target policy will be inaccurate.

Step 2:- For M number of iterations:

1. Generate an episode using policy π
2. Store all rewards obtained in the episode in the list called rewards.
3. For each step t in the episode:

If (s_t, a_t) is occurring for the first time in the episode:

1. Compute the return of a state-action pair,

$$R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$$

2. Update the total return of the state-action pair as $\text{total\_return}(s_t, a_t) = \text{total\_return}(s_t, a_t) + R(s_t, a_t)$.

3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$

4. Compute the Q value by just taking the average, i.e.,

$$Q(s_t, a_t) = \frac{\text{total\_return}(s_t, a_t)}{N(s_t, a_t)}$$

4. Compute the updated policy π using the Q function. Let $a^* = \arg\max_a Q(s, a)$. The policy π selects the best action a^* with probability $1-\epsilon$ and random action with probability ϵ.

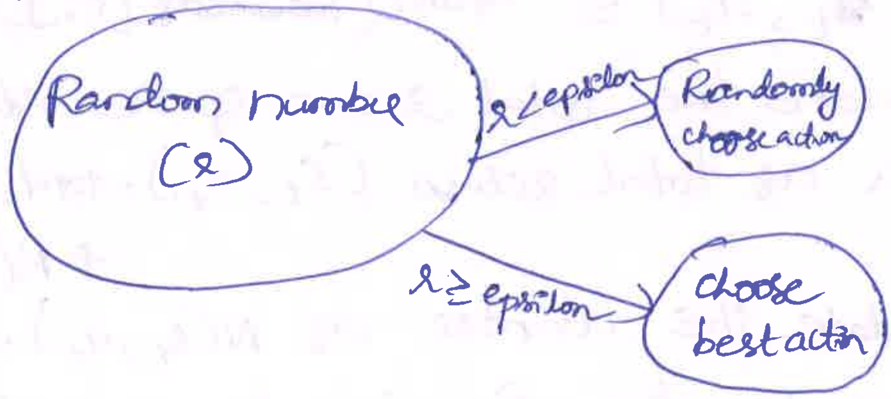
Off-policy Monte Carlo Control

→ behavior policy:- We behave (generate episodes) using the behavior policy.

→ target policy:- we try to improve the other policy called the target policy.

action that has the maximum Q value. i.e., with a probability epsilon, we select a random action (exploration) and with a probability 1 - epsilon we select the best action (exploitation).

epsilon 0 → Exploitation
Epsilon 1 → Exploration.
Optimal → between 0 and 1.



Epsilon-greedy policy

```
def epsilon_greedy_policy (state, epsilon):
    if random.uniform(0, 1) < epsilon:
        return env.action_space.sample()
    else:
        return max(list(range(env.action_space
            .n)), key = lambda x: q[(state, x)]).
```

The MC control algorithm with the epsilon-greedy policy

Step 1:- Let total_return (s, a) be the sum of the return of a state-action pair across several episodes and N(s, a) be the number of times a state-action pair is visited across several episodes. Initialize total_return (s, a) and N(s, a) for all state-action pairs to zero and random policy π.

3. Update the counter as $N(S_t, a_t) = N(S_t, a_t) + 1$

4. Compute the Q value by just taking the average, i.e.,

$$Q(S_t, a_t) = \frac{\text{total\_return}(S_t, a_t)}{N(S_t, a_t)}$$

Step 5:- Compute the updated policy π using the Q function:

$$\pi = \arg\max_a Q(s, a).$$

But, exploring starts method is not applicable to every environment. We can't just randomly chose any state-action pair as an initial state-action pair. To overcome this we have Monte Carlo with the epsilon-greedy policy.

<u>Monte Carlo with the epsilon-greedy policy.</u>

→ Whether the agent should explore all the other actions in the state and select the best action as the one that has the maximum Q value or exploit the best action out of already-explored actions. This is called an <u>exploration-exploitation dilemma.</u>

To avoid this dilemma, we introduce a new policy called the epsilon-greedy policy. Here, all actions are tried with an non-zero probability (epsilon). With a probability epsilon, we explore different actions randomly with a probability 1-epsilon, we chose an

Two types of on-policy Monte Carlo Control methods

⇒ Monte Carlo exploring starts

⇒ Monte Carlo with the epsilon-greedy policy.

Algorithm :-

Step ① Let total_return(s, a) be the sum of the return of a state-action pair across several episodes and N(s, a) be the number of times a state-action pair is visited across several episodes. Initialize total_return(s, a) and N(s, a) for all state action pairs to zero and initialize a random policy π.

Step ② For M number of iterations:

1. Select the initial state s_0 and initial action a_0 randomly such that all state-action pairs have a probability greater than 0.
2. Generate an episode from the selected initial state s_0 and action a_0 using policy π.
3. Store all the rewards obtained in the episode in the list called rewards.
4. For each step t in the episode:

   If (s_t, a_t) is occuring for the first time in the episode:

   1. Compute the return of a state-action pair, $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$
   2. Update the total return of the state-action as total_return (s_t, a_t) = total_return (s_t, a_t) + $R(s_t, a_t)$.

2. Update the total return of the state-action pair as, $total\_return(s_t, a_t) = total\_return(s_t, a_t) + R(s_t, a_t)$.

3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$

4. Compute the Q value by just taking the average, i.e.,

$$Q(s_t, a_t) = \frac{total\_return(s_t, a_t)}{N(s_t, a_t)}$$

Step 4:- Compute the new updated policy π using the Q function:

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

Control methods:-

On-policy control:- the agent behaves using one policy and also tries to improve the same policy. We generate episodes using one policy and also improves the same policy iteratively to find the optimal policy.

Off-policy control:- the agent behaves using one policy b and tries to improve a different policy π. We generate episodes using one policy and we try to improve the different policy iteratively to find the optimal policy.

We repeat this process for several iterations until find the optimal policy π^*.

$$\pi_0 \to Q^{\pi_0} \to \pi_1 \to Q^{\pi_1} \to \pi_2 \to Q^{\pi_2} \to \pi_3 \to Q^{\pi_3} \to \cdots \to \pi^* \to Q^*$$

Fig: Path to find the optimal policy

This step is called policy evaluation and improvement. Policy evaluation implies that at each step we evaluate the policy. Policy improvement implies that at each step we are improving the policy by taking the maximum Q value.

MC control algorithm

Step 1:- Let total_return (s, a) be the sum of the return of a state-action pair across several episodes and N(s, a) be the number of times a state-action pair is visited across several episodes. Initialize total_return (s, a) and N(s, a) for all state-action pairs to zero and initialize a random policy π.

Step 2:- For M number of iterations:

1. Generate an episode using policy π
2. Store all rewards obtained in the episode in the list called rewards
3. For each step t in the episode:

   If (s_t, a_t) is occurring for the first time in the episode:

   1. Compute the return of a state-action pair $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$.

Monte-Carlo Control

→ Here, the goal is to find the optimal policy.
→ We have a Q function, then we can extract policy by selecting an action in each state that has the maximum Q value:

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

Iteration 1 — Let π_0 be the random policy. We use this random policy to generate an episode, and then we compute the Q function Q^{π_0} by taking the average return of the state-action pair. Then, from this Q function Q^{π_0}, we extract a new policy π_1. This new policy π_1 will not be an optimal policy since it is extracted from the Q function, which is computed using the random policy.

Iteration 2 — So, we use the new policy π_1 derived from the previous iteration to generate an episode and compute the new Q function Q^{π_1} as average return of a state-action pair. Then from this Q^{π_1}, extract a new policy π_2, if π_2 is optimal stop, else go to iteration 3.

Iteration 3 — Now, we use the new policy π_2 derived from the previous iteration to generate an episode and compute the new Q function Q^{π_2}, then from Q^{π_2} extract a new policy π_3. If π_3 is optimal stop, else go to the next iteration.

Step 2:- For M number of iterations:

1. Generate an episode using policy π.
2. Store all rewards obtained in the episode in the list called rewards.
3. For each step t in the episode:
   1. Compute return for the state-action pair,
      $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$
   2. Update total return of the state-action pair,
      $\text{total\_return}(s_t, a_t) = \text{total\_return}(s_t, a_t) + R(s_t, a_t)$
   3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$

Step 3:- Compute the Q function (Q value) by just taking the average, i.e., $Q(s,a) = \dfrac{\text{total\_return}(s,a)}{N(s,a)}$

MC prediction of the Q function also has two types

→ first-visit MC:- We compute the return of the state-action pair only for the first time the state-action pair is visited in the episode.

→ every-visit MC:- We compute the return of the state-action pair every time the state-action pair is visited in the episode.

Q value using the incremental mean

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t))$$

MC prediction (Q function)

$$Q(s,a) = \frac{\text{total\_return}(s,a)}{N(s,a)}$$

total_return $(s,a) \rightarrow$ the sum of the return of the state-action pair across several episodes.

$N(s,a) \rightarrow$ the number of times the state-action pair is visited across several episodes.

Eg:-

State	Action	total_return(s,a)	N(s,a)
s_0	0	4	2
s_0	1	2	2
s_1	0	2	2
s_1	1	2	1

$Q(s_0, 0) = \text{total\_return}(s_0, 0) / N(s_0, 0) = 4/2 = 2$

$Q(s_0, 1) = \cancel{\text{total\_return}(s_0, 1)}) / N(s_0, 1) = 2/2 = 1$

$Q(s_1, 0) = 2/2 = 1$

$Q(s_1, 1) = 2/1 = 2$

Algorithm:-

Step 1:- Let total_return (s,a) be the sum of the return of a state-action pair across several episodes and $N(s,a)$ be the number of times a state-action pair is visited across several episodes. Initialize total_return (s,a) and $N(s,a)$ for all state-action pairs to zero. The policy π is given as input.

Incremental mean updates

In both first-visit MC and every-visit MC, we estimate the value of a state as an average (arithmetic mean) return of the state across several episodes

$$V(s) = \frac{total\_return(s)}{N(s)}$$

Incremental mean, $N(S_t) = N(S_t) + 1$

$$V(S_t) = V(S_t) + \frac{1}{N(S_t)} (R_t - V(S_t))$$

If the environment is non-stationary we can ignore returns from earlier episodes and use only the returns from the latest episodes for computing the average.

$$V(S_t) = V(S_t) + \alpha (R_t - V(S_t))$$

where $\alpha = \frac{1}{N(S_t)}$ and R_t is the return of the state S_t.

After second iteration:

State	total_return(S)	N(S)
S_0	6	2
S_1	2	2

Step 3:-

$$V(S) = \frac{\text{total\_return}(S)}{N(S)}$$

$$V(S_0) = \frac{\text{total\_return}(S_0)}{N(S_0)} = \frac{6}{2} = 3$$

$$V(S_1) = \frac{\text{total\_return}(S_1)}{N(S_1)} = \frac{2}{2} = 1$$

Types of MC Prediction

→ First-visit Monte Carlo

→ Every-visit Monte Carlo.

Algorithm for the above types remains same as MC prediction.

First-visit Monte Carlo → if the same state is visited again in the same episode, we don't compute the return for that state again.

Every-visit Monte Carlo → We compute the return every time a state is visited in the episode.

$$R(S_1) = \text{Sum}(\text{rewards}[1:])$$
$$= \text{Sum}([1])$$
$$= 1$$

$$\text{total\_returns}(S_1) = \text{total\_returns}(S_1) + 1$$
$$= 0 + 1 = 1$$
$$N(S_1) = N(S_1) + 1$$
$$= 0 + 1 = 1$$

After iteration 1

State	total_return(S)	N(S)
S_0	2	1
S_1	1	1

Iteration 2

$$S_0 \xrightarrow[+3]{a=1} S_1 \xrightarrow[+1]{a=0} S_2$$

$$\text{rewards} = [3, 1]$$
$$R(S_0) = \text{Sum}(\text{Rewards}[0:])$$
$$= \text{Sum}([3, 1])$$
$$= 4$$
$$\text{total\_returns}(S_0) = \text{total\_returns}(S_0) + R(S_0)$$
$$= 2 + 4 = 6$$
$$N(S_0) = N(S_0) + 1 = 1 + 1 = 2$$

$$R(S_1) = \text{Sum}(\text{rewards}[1:])$$
$$= \text{Sum}([1])$$
$$= 1$$
$$\text{total\_returns}(S_1) = \text{total\_returns}(S_1) + R(S_1)$$
$$= 1 + 1 = 2$$
$$N(S_1) = N(S_1) + 1 = 1 + 1 = 2$$

(2) For M number of iterations:

1. Generate an episode using the policy π
2. Store all the rewards obtained in the episode in the list called rewards
3. For each step t in the episode:
   1. compute the return of state S_t as
      $R(S_t) = \text{Sum}(\text{rewards}[t:])$
   2. Update the total return of state S_t as
      $\text{total\_returns}(S_t) = \text{total\_return}(S_t) + R(S_t)$
   3. Update the counter as $N(S_t) = N(S_t) + 1$

(3) Compute the value of a state by just taking the average, i.e.,

$$V(s) = \frac{\text{total\_return}(s)}{N(s)}$$

Eg:- To compute the value of three states S_0, S_1 & S_2

Step1:

State	total_return(s)	N(s)
S_0	0	0
S_1	0	0

Step2:

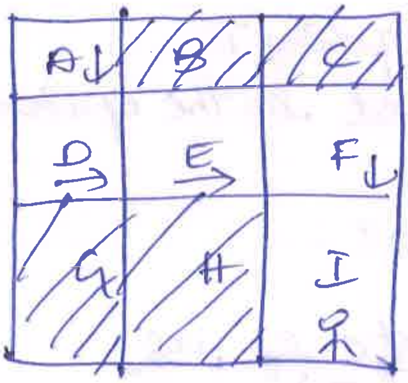
① $$S_0 \xrightarrow[+1]{a=0} S_1 \xrightarrow[+1]{a=1} S_2$$

② rewards = [1, 1]

③ $R(S_0) = \text{Sum}(\text{rewards}[0:])$
$= \text{Sum}([1, 1])$
$= 2$

$\text{total\_returns}(S_0) = \text{total\_returns}(S_0) + R(S_0)$
$= 0 + 2$
$= 2$

$N(S_0) = N(S_0) + 1 = 0 + 1 = 1$



$$T_3 = (A) \xrightarrow[+1]{Do} (D) \xrightarrow[+1]{Ri} (E) \xrightarrow[+1]{Ri} (F) \xrightarrow[+1]{Do} (I)$$

$$R_3(A) = 1+1+1+1 = 4$$

$$V(S) \simeq \frac{1}{N}\sum_{i=1}^{N} R_i(S)$$

$$V(A) \simeq \frac{1}{N}\sum_{i=1}^{N} R_i(A)$$

$$V(A) \simeq \frac{1}{3}\sum_{i=1}^{3} R_i(A)$$

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

$$= \frac{1}{3}\left[10\right]$$

$$= 3.3$$

$|||^{ly}$, we can compute the value of all other states by just taking the average return of the state across the three episodes.

MC prediction algorithm

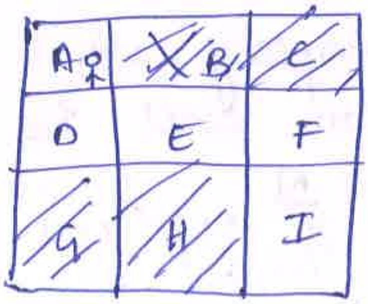
① Let total_return (s) be the sum of return of a state across several episodes and N(s) be the counter, i.e., the number of times a state is visited across several episodes. Initialize total_return (s) and N(s) as zero for all the states. The policy π is given as input.

So, we will start off by initializing a random policy and we try to find the optimal policy iteratively. That is, we try to find an optimal policy that gives the maximum return.

Monte Carlo Prediction

→ We approximate the value of a state by taking the average return of a state across N episodes instead of taking the expected return.

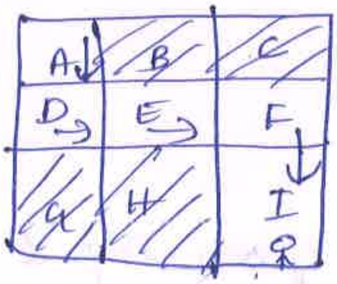
Eg:-



$$V(S) \simeq \frac{1}{N}\sum_{i=1}^{N} R_i(S)$$

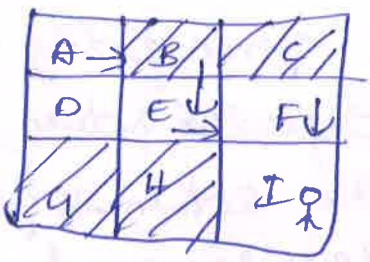
Grid world Environment

Let's generate an episode τ_1 using our given stochastic policy π.



$$t_1 = (A) \xrightarrow[+1]{\text{Down}} (D) \xrightarrow[+1]{\text{Right}} (E) \xrightarrow[+1]{\text{Right}} (F) \xrightarrow[+1]{\text{Down}} (I)$$

$$R_1(A) = 1+1+1+1 = 4$$



$$t_2 = (A) \xrightarrow[-1]{\text{Ri}} (B) \xrightarrow[+1]{\text{Do}} (E) \xrightarrow[+1]{\text{Ri}} (F) \xrightarrow[+1]{\text{Do}} (I)$$

$$R_2(A) = -1+1+1+1 = 2$$

Monte Carlo Methods → Chapter 4 (Module 3)

→ Model-free methods do not require the model dynamics of the environment to compute the value and Q functions in order to find the optimal policy.

→ Monte Carlo (MC) method is one of the model-free methods.

→ Monte Carlo method approximates the expectation of a random variable by sampling, when the sample size is greater, the approximation will be better.

$$E(x) = \sum_{i=1}^{N} x_i \, p(x_i)$$

$$E_{x\sim p(x)}[X] = \frac{1}{N} \sum_i x_i$$

→ Two important tasks in reinforcement learning:

→ The prediction task
→ The control task

Prediction task

→ We don't make any change to the given input policy. We keep the given policy as fixed and predict the value function or Q function using the given policy and the expected return. Based on the expected return, we can evaluate the given policy. If the return is good then we can say that the given policy is good.

Control task

→ Unlike the prediction task, in control task, we will not be given any policy as an input. In the control task our goal is to find the optimal policy.