Monte Carlo Methods

To use a pure dynamic programming approach, the agent must know the transition probabilities. However,

- In some problems such as weather forecasting, it is impossible to know the probability beforehand.
- The computation process can be error-prone and tedious as the value range gets larger

The Monte Carlo method can help estimate the value by averaging over a large number of random samples to get the most accurate results to the expected. In short, it is simply about estimation replies on repeated random sampling.

MC Prediction for estimating $V \approx v_\pi$

```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Return(s) \leftarrow$ an empty list for all $s \in \mathcal{S}$
Loop forever (for each episode):
    Generate an episode using $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \dots, 0$:
        $G \leftarrow \gamma G + R_{T+1}$
        Append G to $Return(S_t)$
        $V(S_t) \leftarrow average\big(Return(S_t)\big)$
```

Using Monte Carlo for Action values

To determine the action value, we average the returns following by a policy.

$$q_\pi(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

Action value allows us to compare all different actions in the same state and select the best possible action. However, this is only possible if we have an estimate of the values of the other actions.

MC 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 $s \in \mathcal{S}, a \in \mathcal{A}(s)$
        $Return(s, a) \leftarrow$ an 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 $S_0, A_0$ using $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
        $G \leftarrow 0$
        Loop for each step of episode, $t = T - 1, T - 2, \ldots, 0$:
                $G \leftarrow \gamma G + R_{T+1}$
                Append G to $Returns(S_t, A_t)$
                $Q(S_t, A_t) \leftarrow average\big(Returns(S_t, A_t)\big)$
                $V(S_t) \leftarrow average\big(Returns(S_t, A_t)\big)$
```

Epsilon-soft policies

The drawback of exploring start in Monte Carlo algorithms is It must be able to start from every possible state-action pair, otherwise not explore enough and could converge to a suboptimal solution.

Epsilon-soft policies take each action with probability at least ϵ over the number of actions. These policies force the agent to continually explore which eliminates the need of exploring start.

The ϵ-soft policy assigns non-zero probability to every action of every state. Because of this, agents continue to visit all state action pairs indefinitely.

ϵ-soft policies are always stochastic, which specifies the probability of taking action of each state in ϵ.

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

```
Initialize:
        $\pi(s) \leftarrow$an arbitrarily $\epsilon$-soft policy
        $Q(s, a) \in \mathbb{R}$ , arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
        $Return(s, a) \leftarrow$ an empty list for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Loop forever (for each episode):
        Generate an episode from $S_0, A_0$ using $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
        $G \leftarrow 0$
```

```
Loop for each step of episode, $t = T-1, T-2, \dots, 0$:
    $G \leftarrow \gamma G + R_{T+1}$
    Append G to $Returns(S_t, A_t)$
    $Q(S_t, A_t) \leftarrow average\big(Returns(S_t, A_t)\big)$
    $A^* \leftarrow argmax_a Q(S_t, a)$
    For all $a \in \mathcal{A}(S_t)$:
        $\pi(a|S_t) \leftarrow \begin{cases} 1-\epsilon+\frac{\epsilon}{|\mathcal{A}(S_t|} & if\ a = A^* \\ \frac{\epsilon}{|\mathcal{A}(S_t|} & if\ a \neq A^* \end{cases}$
```

Off-policy learning

ϵ-soft policies helps with the continuing exploring problems by adding the probability for every action on each time step. However, their disadvantage is they are sub-optimal for both acting and learning

1. **On-policy and Off-policy learning**
    - On-policy: improve and evaluate the policy being used to select actions
    - Off-policy: improve and evaluate a different policy from the one used to select actions.

2. **Continual exploration**

Target policy	Behavior policy
$\pi(a\|s)$ • Learn values for this policy • For example the optimal policy	$b(a\|s)$ • Selecting actions from his policy • Generally an exploratory policy

There are some applications for off-policy learning:
- <u>Facilitating exploration</u>
- Learning from demonstration
- Parallel learning

3. **Key rule of off-policy learning**

The behavior must cover the target policy

$$\pi(a|s) > 0\ where\ b(a|s) > 0$$

Importance Sampling

$$\begin{aligned}\mathbb{E}_\pi[X] &\doteq \sum_{x\in X} x\pi(x)\\ &= \sum_{x\in X} x\pi(x)\frac{b(x)}{b(x)}\\ &= \sum_{x\in X} x\frac{\pi(x)}{b(x)}b(x)\\ &= \sum_{x\in X} x\rho(x)b(x)\end{aligned}$$

$$\begin{aligned}\mathbb{E}_\pi[X] &= \sum_{x\in X} x\rho(x)b(x)\\ &= \mathbb{E}_b[X\rho(X)]\end{aligned}$$

$$\begin{aligned}\mathbb{E}_b[X\rho(X)] &= \sum_{x\in X} x\rho(x)b(x)\\ &\approx \frac{1}{n}\sum_{i=1}^{n} x_i\rho(x_i)\end{aligned}$$