Reinforcement Learning (3)

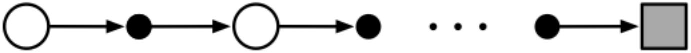
Maxime Berar

16 octobre 2025

Monte-Carlo Methods

- MC learning through complete episodes, model-free

Episode : a state-reward sequence



Complete return of a state in an episode

$$G_t(e) \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

is used to estimate v_π

$$v_\pi(s) = \frac{1}{n^s_{\text{episodes}}} \sum_{e=1}^{n^s_{\text{episodes}}} G_t(e)$$

- 2 versions : first-visit *or* every-visits

Algorithm 1 First-visit MC prediction, for estimating $V \approx v_\pi$

```
1: Initialize:
2: $\pi \leftarrow$ policy to be evaluated
3: $V \leftarrow$ an arbitrary state-value function
4: $Returns(s) \leftarrow$ an empty list, $\forall s \in \mathcal{S}$
5: while forever do
6:     Generate an episode using $\pi$
7:     for each state $s$ appearing in the episode do
8:         $G \leftarrow$ the return that follows the first occurrence of $s$
9:         Append $G$ to $Returns(s)$
10:        $V(s) \leftarrow$ average$(Returns(s))$
11:    end for
12: end while
```

- The first-visit MC method estimates $v_\pi(s)$ as the average of the returns following first visits to s. It dates back to the 1940s.

Visits

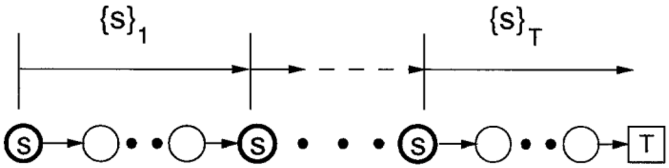
- The first-visit MC ...
- The every-visit MC method averages the returns following all visits to s. Every-visit MC extends more naturally to function approximation and eligibility traces (as presented later).

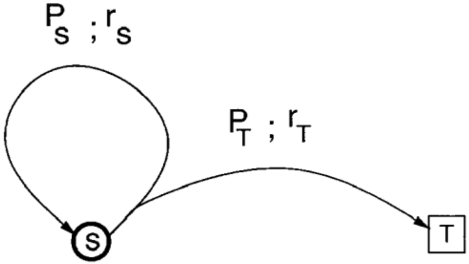
Both algorithms converge to $v_\pi(s)$ as the number of (first) visits to s goes to infinity. In the case of first-visit MC,

$$\hat{v}_\pi(s) = \frac{1}{n_{\text{ep}}^s} \sum_{e=1}^{n_{\text{ep}}^s} G_t(e)$$

each return $G_t(e)$ is an iid estimate of $v_\pi(s)$ with finite variance. The sequence of averages of these estimates converges to their expected value. Each average is itself an unbiased estimate, and the standard deviation of its error falls as $1/n_{\text{ep}}^s$, where n_{ep}^s is the number of returns averaged. The samples obtained by the every-visit strategy are dependent so they're not i.i.d. samples compared to first-visit strategy.

Reduction to a Two-State Abstracted Markov Chain





Singh and Sutton, *Reinforcement Learning with Replacing Eligibility Traces*, Machine Learning 22 (1996)

First Visit MC

Let $\{x\}$ stand for the paired random sequence $(\{s\},\{r\})$. The first-visit MC estimate for $V(s)$ after one trial, $\{x\}$, is

$$V_1^F(s) = f(\{x\}) = r_{s_1} + r_{s_2} + \cdots + r_{s_k} + r_T,$$

where k is the random number of revisits to state s, r_{s_i} is the sum of the individual rewards in the sequence $\{r\}_i$, and r_T is the random total reward received after the last visit to state s.
For all i, $\mathbb{E}[r_s] = R_s$. The first-visit MC estimate of $V(s)$ after n trials, $\{x\}^1, \{x\}^2, \cdots, \{x\}^n$, is

$$V_n^F(s) = f(\{x\}^1, \cdots, \{x\}^n) = \frac{\sum_{i=1}^n f((\{x\}^i)}{n}$$

Every visit MC

$$V_1^E(s) = t(\{x\}) = \frac{t_{num}(\{x\})}{k+1} = \frac{r_{s_1} + 2r_{s_2} + \cdots + kr_{s_k} + (k+1)r_T}{k+1},$$

where k is the random number of revisits to state s in the sequence $\{x\}$. Every visit to state s effectively starts another trial. The every-visit MC estimate after n trials,$\{x\}^1, \{x\}^2, \cdots, \{x\}^n$, is

$$V_n^E(s) = t(\{x\}^1, \cdots, \{x\}^n) = \sum_{i=1}^{n} \frac{t_{num,i}(\{x\})}{k_i + 1}$$

Some results on First Visit

From Bellman's equation

$$V(s) = \frac{P_s}{P_T} R_S + R_T$$

Expectation of $V_1^F(s)$

$$\begin{aligned}
\mathbb{E}\left[f(\{x\})\right] = \sum_{\{x\}} P(\{x\}) f(x) &= \sum_k P(k) \mathbb{E}_r\left[f(\{x\})|k\right], \\
&= \sum_k P_T P_s^k (k R_S + R_T) \\
&= P_T \left(\frac{P_s}{1 - P_s^2} R_s + \frac{1}{1 - P_s} R_T \right) = V(s)
\end{aligned}$$

Every-Visit MC is Biased

Expectation of $V_1^E(s)$

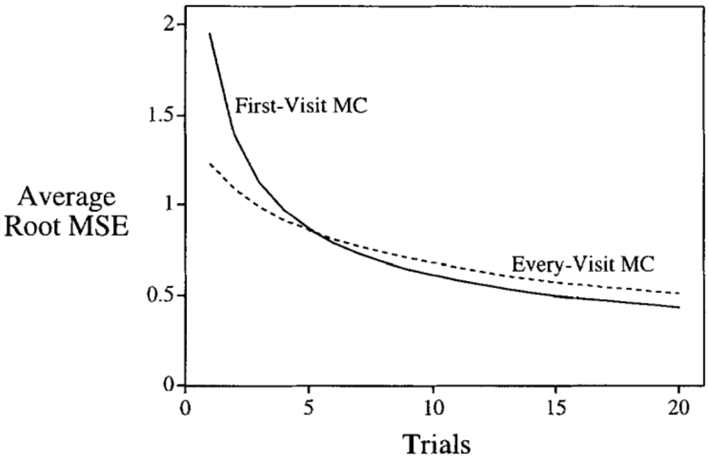
$$\begin{aligned}
\mathbb{E}\left[t(\{x\})\right] &= \sum_{\{x\}} P(\{x\}) t(x) \\
&= \sum_{k} P(k) \mathbb{E}_r\left[t(\{x\}) | k\right], \\
&= \sum_{k} P_T P_s^k \left(\frac{R_s + 2R_s + \cdots + kR_s + (k+1)R_T}{k+1}\right) \\
&= \sum_{k} P_T P_s^k \left(\frac{k}{2} R_s + R_T\right) \\
&= \frac{P_s}{2P_T} R_S + R_T \neq V(s)
\end{aligned}$$

After n trials

$$\mathbb{E}\left[V_n^E(s)\right] = \frac{nP_s}{(n+1)P_T} R_S + R_T$$

every-visit MC algorithm is unbiased in the limit $n \to \infty$

Comparison


First-Visit MC
Every-Visit MC
Average Root MSE
Trials
0
0.5
1
1.5
2
0
5
10
15
20

[illegible]

Monte Carlo Estimation of Action Values

- without a model, state values alone are not sufficient.
- useful to estimate action values (the values of state-action pairs)

The policy evaluation problem for action values is to estimate $q_\pi(s, a)$, the expected return when starting in state s, taking action a, and thereafter following policy π.

- The MC methods for this are essentially the same as just presented for state values,visits to a state-action pair rather than to a state
- first-visit MC *vs* every-visit MC

Pb ! complication is that many state-action pairs may never be visited

- for policy evaluation to work for action values, we must assure continual exploration.
- One way to do this is by specifying that the episodes start in a state-action pair, and that every pair has a nonzero probability of being selected as the start.

the assumption of exploring starts.

alternative : consider only policies that are stochastic with a nonzero probability of selecting all actions in each state.

Monte Carlo Control

How Monte Carlo estimation can be used to approximate optimal policies ?

- Generalized Policy Iteration : one maintains both an approximate policy and an approximate value function.
- The value function is repeatedly altered to more closely approximate the value function for the current policy, and the policy is repeatedly improved with respect to the current value function.
- These 2 improvement kinds of changes work against each other to some extent, but together they cause both policy and value function to approach optimality.

2 theorical assumptions

- the episodes have exploring starts,
- policy evaluation could be done with an infinite number of episodes

Policy evaluation is done exactly as described before.
Policy improvement is done by making the policy greedy with respect to the current value function.

Algorithm 2 Monte Carlo Exploring Starts, for estimating $\pi \approx \pi^*$

```
 1: Initialize, $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ :
 2: $Q(s, a) \leftarrow$ arbitrary
 3: $\pi(s) \leftarrow$ arbitrary
 4: $Returns(s, a) \leftarrow$ empty list
 5: while forever do
 6:     Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ $(\Pr(s, a) > 0)$
 7:     Generate an episode starting from $S_0, A_0$, following $\pi$
 8:     for  each pair $s$, $a$ appearing in the episode do
 9:         $G \leftarrow$ the return that follows the first occurrence of $s, a$
10:         Append $G$ to $Returns(s, a)$
11:         $Q(s, a) \leftarrow average(Returns(s, a))$
12:     end for
13:     for each $s$ in the episode do
14:         $\pi(s) \leftarrow \arg\max_a Q(s, a)$
15:     end for
16: end while
```

Monte Carlo Control without Exploring Start

How can we avoid the unlikely assumption of exploring starts ?
2 approaches : on-policy methods and off-policy methods

- On-policy methods attempt to evaluate or improve the policy that is used to make decisions
- Off-policy methods evaluate or improve a policy different from that used to generate the data

In on-policy control methods the policy is generally soft, meaning that $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}(s)$, but gradually shifted closer and closer to a deterministic optimal policy.

Example : ε-greedy policies (cf bandits)

Algorithm 3 On-policy first-visit MC control (for ε-soft policies)

```
1:  for for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ do
2:      $Q(s,a) \leftarrow$ arbitrary
3:      $Returns(s,a) \leftarrow$ empty list
4:  end for
5:  $\pi(a|s) \leftarrow$ an arbitrary $\varepsilon-$soft policy
6:  while forever do
7:      Generate an episode using $\pi$
8:      for each pair $s, a$ appearing in the episode do
9:          $G \leftarrow$ the return that follows the first occurrence of $s, a$
10:         Append $G$ to $Returns(s,a)$
11:         $Q(s,a) \leftarrow$ average$(Returns(s,a))$
12:     end for
13:     for each $s$ in the episode do
14:         $A^* \leftarrow \arg\max_a Q(s,a)$      (with ties broken arbitrarily)
15:         for all $a \in \mathcal{A}(s)$ do
16:             $\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if} \quad a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if} \quad a \neq A^* \end{cases}$
17:         end for
18:     end for
```

Off-policy MC

In off-policy methods these two functions are separated :

- The policy used to generate behavior, called the behavior policy, may in fact be unrelated to the policy that is evaluated and improved, called the target policy.
- An advantage of this separation is that the target policy may be deterministic (e.g., greedy), while the behavior policy can continue to sample all possible actions.
- require that the behavior policy has a nonzero probability of selecting all actions that might be selected by the target policy (coverage).

A potential problem is that this method learns only from the tails of episodes, when all of the remaining actions in the episode are greedy. If nongreedy actions are common, then learning will be slow, particularly for states appearing in the early portions of long episodes.

Algorithm 4 Off-policy MC prediction, for estimating $Q \approx q_\pi$

```
INPUT : an arbitrary target policy $\pi$
**for** all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ **do**
    $Q(s, a) \leftarrow$ arbitrary
    $C(s, a) \leftarrow 0$
**end for**
**while** forever **do**
    $b \leftarrow$ any policy with coverage of $\pi$
    Generate an episode using $b$
    $G \leftarrow 0$
    $W \leftarrow 1$
    **for** $t = T - 1, T - 2, \cdots$ down to 0 **do**
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        If $W = 0$ then exit For loop
    **end for**
**end while**
```

Algorithm 5 Off-policy MC control, for estimating $\pi \approx \pi^*$

```
for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ do
    $Q(s, a) \leftarrow$ arbitrary
    $C(s, a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(s, a)$    (with ties broken consistently)
end for
while forever do
    $b \leftarrow$ any soft policy
    Generate an episode using $b$
    $G \leftarrow 0$
    $W \leftarrow 1$
    for $t = T - 1, T - 2, \cdots$ down to 0 do
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)}[G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit For loop
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
    end for
end while
```

Let's Play BlackJack

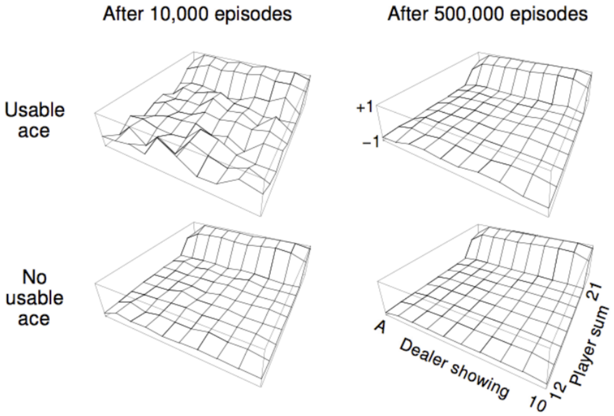
- 1 player (the agent) vs. 1 dealer
- Card values :
    - 10 for a face card (jack,queen,king),
    - 1 or 11 for an Ace (11 for the first encounter then 1),
    - Value on the card for any card between 2 and 9.
- Initialisation : 2 cards each, one is face down for the dealer
- Game : the player can request as many cards, one by one, as he wishes, until he either stop and exceed 21
- If the player has 21, he wins except if the dealer has 21 too.
- Player goal : score $\geq$ dealer, after both stops .
- Hypothesis : infinite deck

MDP for BlackJack

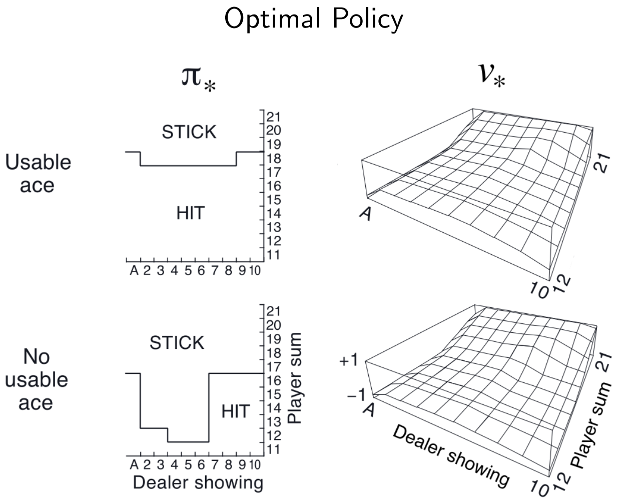
- 200 states :
  - player current sum : $(12 - 21)$ [before hit]
  - Value of the face up card of the dealer : $(1 - 10)$
  - *Ace* already dealed (yes-no)
- 2 actions *hit* ou *stick*
- Reward :
  - 1 if win
  - -1 si lost (less than dealer or > 21)
  - 0 if null or continue
- $\gamma = 1$
- Dealer policy : *hit* if total sum ≤ 17

MC methods for BlackJack

Policy to be evaluated : stick if score $\geq$ 20.



MC methods for BlackJack



Temporal Difference

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.

- learning with incomplete episodes
- model-free
- updating estimations with estimations

update from every-visit Monte Carlo method

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right]$$

Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$. TD(0) will try to do the same with only one step of time

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\hat{G}_t} - V(S_t) \right]$$

Algorithm 6 Algorithm TD(0) for estimating v_π

```
1:  $v(s) \leftarrow 0 \forall s \in \mathcal{S}$
2:  **repeat**
3:      initialize $S$
4:      **repeat**
5:          $A \leftarrow$ action given by $\pi$ for $s$
6:          Take action $A$, observe $R$ et $s'$
7:          $v(s) \leftarrow v(s) + \alpha \left[R + \gamma v(s') - v(s)\right]$
8:          $s \leftarrow s'$
9:      **until** $s$ is terminal
10:
11: **until** all episodes are done
```

Advantages of TD Prediction Methods

- they do not require a model of the environment, of its reward and next-state probability distributions.
- they are naturally implemented in an on-line, fully incremental fashion.
- some Monte Carlo methods must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning. TD methods are much less susceptible to these problems because they learn from each transition regardless of what subsequent actions are taken.

For any fixed policy π, TD(0) has been proved to converge to v_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions.

Algorithm 7 Algorithm SARSA, on-policy TD control

```
 1: initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily, and
    $Q(\text{terminal-state}, \cdot) = 0$
 2: repeat
 3:     Initialize $S$
 4:     Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
 5:     repeat
 6:         Take action $A$, observe $R$, $S'$
 7:         Choose $A'$ from $S'$ using policy derived from $Q$
 8:         $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$
 9:         $S \leftarrow S'$, $A \leftarrow A'$
10:     until S is terminal
11: until
```

Algorithm 8 Algorithm Q-Learning, off-policy TD control

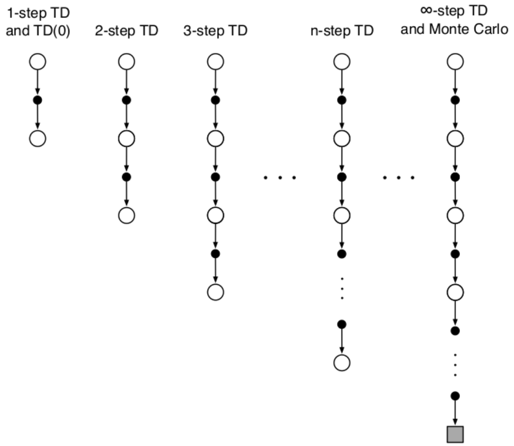
```
 1: initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily, and
    $Q(\text{terminal-state}, \cdot) = 0$
 2: while 1 do
 3:     Initialize $S$
 4:     $t \leftarrow 0$
 5:     repeat
 6:         Choose $A$ from $S$ using policy derived from $Q$
 7:         Take action $A$, observe $R$,$S'$
 8:         $Q(S,A) \leftarrow Q(S,A) + \alpha\,[R + \gamma\,\text{max}_{a' \in \mathcal{A}}\,Q(S',A') - Q(S,A)]$
 9:     until $S$ is terminal
10: end while
```

n-step Bootstrapping

unify the Monte Carlo (MC) methods and the one-step temporal-difference (TD) methods.

- free from the tyranny of the time step.
- a length of time in which a significant and recognizable state change has occurred.

The idea of n-step methods is usually used as an introduction to the algorithmic idea of eligibility traces, which enable bootstrapping over multiple time intervals simultaneously.



$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

n-step TD for estimating $V \approx v_\pi$

```
Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer $n$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n$

Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take an action according to $\pi(\cdot|S_t)$
    |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    |       If $S_{t+1}$ is terminal, then $T \leftarrow t + 1$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose state's estimate is being updated)
    |   If $\tau \geq 0$:
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$                    $(G_{\tau:\tau+n})$
    |       $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
    Until $\tau = T - 1$
```

n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

```
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$, or to a fixed given policy
Parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n$

Repeat (for each episode):
   Initialize and store $S_0 \neq$ terminal
   Select and store an action $A_0 \sim \pi(\cdot|S_0)$
   $T \leftarrow \infty$
   For $t = 0, 1, 2, \ldots$ :
   |  If $t < T$, then:
   |     Take action $A_t$
   |     Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
   |     If $S_{t+1}$ is terminal, then:
   |        $T \leftarrow t+1$
   |     else:
   |        Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
   |  $\tau \leftarrow t-n+1$    ($\tau$ is the time whose estimate is being updated)
   |  If $\tau \geq 0$:
   |     $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
   |     If $\tau+n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$          $(G_{\tau:\tau+n})$
   |     $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
   |     If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
   Until $\tau = T-1$
```

Off-policy n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

```
Input: an arbitrary behavior policy $b$ such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$, or as a fixed given policy
Parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n$

Repeat (for each episode):
   Initialize and store $S_0 \neq$ terminal
   Select and store an action $A_0 \sim b(\cdot|S_0)$
   $T \leftarrow \infty$
   For $t = 0, 1, 2, \ldots$ :
   |   If $t < T$, then:
   |       Take action $A_t$
   |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
   |       If $S_{t+1}$ is terminal, then:
   |           $T \leftarrow t + 1$
   |       else:
   |           Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
   |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose estimate is being updated)
   |   If $\tau \geq 0$:
   |       $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$                    $(\rho_{\tau+1:t+n-1})$
   |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
   |       If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$          $(G_{\tau:\tau+n})$
   |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$
   |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
   Until $\tau = T - 1$
```