Temporal Difference Methods, Off-Policy Methods

Milan Straka

October 19, 2020
Recall that a Markov decision process (MDP) is a quadruple \((S, A, p, \gamma)\), where:

- \(S\) is a set of states,
- \(A\) is a set of actions,
- \(p(S_{t+1} = s' \mid S_t = s, A_t = a)\) is a probability that action \(a \in A\) will lead from state \(s \in S\) to \(s' \in S\), producing a reward \(r \in \mathbb{R}\),
- \(\gamma \in [0, 1]\) is a discount factor.

Partially observable Markov decision process extends the Markov decision process to a sextuple \((S, A, p, \gamma, O, o)\), where in addition to an MDP

- \(O\) is a set of observations,
- \(o(O_t \mid S_t, A_{t-1})\) is an observation model, which is used as agent input instead of \(S_t\).

Although planning in general POMDP is undecidable, several approaches are used to handle POMDPs in robotics (to model uncertainty, imprecise mechanisms and inaccurate sensors, ...). In deep RL, partially observable MDPs are usually handled using recurrent networks, which model the latent states \(S_t\).
A **policy** $\pi$ computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action $a$ in state $s$.

To evaluate a quality of a policy, we define **value function** $v_\pi(s)$, or **state-value function**, as

$$v_\pi(s) \overset{\text{def}}{=} \mathbb{E}_\pi \left[ G_t | S_t = s \right] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right].$$

An **action-value function** for a policy $\pi$ is defined analogously as

$$q_\pi(s, a) \overset{\text{def}}{=} \mathbb{E}_\pi \left[ G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right].$$

Optimal state-value function is defined as $v_*(s) \overset{\text{def}}{=} \max_\pi v_\pi(s)$, analogously optimal action-value function is defined as $q_*(s, a) \overset{\text{def}}{=} \max_\pi q_\pi(s, a)$.

Any policy $\pi_*$ with $v_{\pi_*} = v_*$ is called an **optimal policy**.
Optimal value function can be computed by repetitive application of Bellman optimality equation:

\[ v_0(s) \leftarrow 0 \]
\[ v_{k+1}(s) \leftarrow \max_a \mathbb{E} [R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] = Bv_k. \]

Converges for finite-horizon tasks or when discount factor \( \gamma < 1 \).
Policy iteration consists of repeatedly performing policy evaluation and policy improvement:

\[
\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} v_{\pi^*}.
\]

The result is a sequence of monotonically improving policies \( \pi_i \). Note that when \( \pi' = \pi \), also \( v_{\pi'} = v_{\pi} \), which means Bellman optimality equation is fulfilled and both \( v_\pi \) and \( \pi \) are optimal.

Considering that there is only a finite number of policies, the optimal policy and optimal value function can be computed in finite time (contrary to value iteration, where the convergence is only asymptotic).

Note that when evaluating policy \( \pi_{k+1} \), we usually start with \( v_{\pi_k} \), which is assumed to be a good approximation to \( v_{\pi_{k+1}} \).
Generalized Policy Evaluation is a general idea of interleaving policy evaluation and policy improvement at various granularity.

If both processes stabilize, we know we have obtained optimal policy.
Monte Carlo methods are based on estimating returns from complete episodes. Furthermore, if the model (of the environment) is not known, we need to estimate returns for the action-value function $q$ instead of $v$.

We can formulate Monte Carlo methods in the generalized policy improvement framework. Keeping estimated returns for the action-value function, we perform policy evaluation by sampling one episode according to current policy. We then update the action-value function by averaging over the observed returns, including the currently sampled episode.

We considered two variants of exploration:

- exploring starts
- $\epsilon$-soft policies
On-policy every-visit Monte Carlo for \( \varepsilon \)-soft Policies

Algorithm parameter: small \( \varepsilon > 0 \)

Initialize \( Q(s, a) \in \mathbb{R} \) arbitrarily (usually to 0), for all \( s \in S, a \in A \)

Initialize \( C(s, a) \in \mathbb{Z} \) to 0, for all \( s \in S, a \in A \)

Repeat forever (for each episode):

- Generate an episode \( S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \), by generating actions as follows:
  - With probability \( \varepsilon \), generate a random uniform action
  - Otherwise, set \( A_t \overset{\text{def}}{=} \arg \max_a Q(S_t, a) \)

- \( G \leftarrow 0 \)

- For each \( t = T - 1, T - 2, \ldots, 0 \):
  - \( G \leftarrow \gamma G + R_{T+1} \)
  - \( C(S_t, A_t) \leftarrow C(S_t, A_t) + 1 \)
  - \( Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)} (G - Q(S_t, A_t)) \)
The reason we estimate action-value function $q$ is that the policy is defined as

$$\pi(s) \overset{\text{def}}{=} \arg \max_a q_{\pi}(s, a)$$

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

and the latter form might be impossible to evaluate if we do not have the model of the environment.

However, if the environment is known, it is often better to estimate returns only for states, because there can be substantially less states than state-action pairs. 
Temporal-difference methods estimate action-value returns using one iteration of Bellman equation instead of complete episode return.

Compared to Monte Carlo method with constant learning rate $\alpha$, which performs

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

the simplest temporal-difference method computes the following:

$$v(S_t) \leftarrow v(S_t) + \alpha \left[ R_{t+1} + \gamma v(S_{t+1}) - v(S_t) \right],$$
TD Methods

<table>
<thead>
<tr>
<th>State</th>
<th>Elapsed Time (minutes)</th>
<th>Predicted Time to Go</th>
<th>Predicted Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaving office, friday at 6</td>
<td>0</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>5</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>exiting highway</td>
<td>20</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>2ndary road, behind truck</td>
<td>30</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>entering home street</td>
<td>40</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>arrive home</td>
<td>43</td>
<td>0</td>
<td>43</td>
</tr>
</tbody>
</table>

Example 6.1 of "Reinforcement Learning: An Introduction, Second Edition".

Figure 6.1 of "Reinforcement Learning: An Introduction, Second Edition".
As with Monte Carlo methods, for a fixed policy $\pi$, TD methods converge to $v_\pi$.

On stochastic tasks, TD methods usually converge to $v_\pi$ faster than constant-$\alpha$ MC methods.

Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".
Optimality of MC and TD Methods

Example 6.4 of "Reinforcement Learning: An Introduction, Second Edition".

For state B, 6 out of 8 times return from B was 1 and 0 otherwise. Therefore, $v(B) = \frac{3}{4}$.

- [TD] For state A, in all cases it transferred to B. Therefore, $v(A)$ could be $\frac{3}{4}$.
- [MC] For state A, in all cases it generated return 0. Therefore, $v(A)$ could be 0.

MC minimizes error on training data, TD minimizes MLE error for the Markov process.
A straightforward application to the temporal-difference policy evaluation is Sarsa algorithm, which after generating $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ computes

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t) \right].$$

### Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in S$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
- Initialize $S$
- Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)

Loop for each step of episode:
- Take action $A$, observe $R$, $S'$
- Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma Q(S', A') - Q(S, A) \right]$$

$S \leftarrow S'$; $A \leftarrow A'$;
until $S$ is terminal

MC methods cannot be easily used, because an episode might not terminate if current policy caused the agent to stay in the same state.
Q-learning was an important early breakthrough in reinforcement learning (Watkins, 1989).

\[ q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left( R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t) \right) . \]

**Q-learning (off-policy TD control) for estimating \( \pi \approx \pi_* \)**

- Algorithm parameters: step size \( \alpha \in (0, 1] \), small \( \varepsilon > 0 \)
- Initialize \( Q(s, a) \), for all \( s \in S, a \in A(s) \), arbitrarily except that \( Q(terminal, \cdot) = 0 \)

Loop for each episode:
  - Initialize \( S \)
  - Loop for each step of episode:
    - Choose \( A \) from \( S \) using policy derived from \( Q \) (e.g., \( \varepsilon \)-greedy)
    - Take action \( A \), observe \( R, S' \)
    - \( Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_a Q(S', a) - Q(S, A) \right) \)
  - \( S \leftarrow S' \)
  - until \( S \) is terminal

*Modification of Algorithm 6.5 of “Reinforcement Learning: An Introduction, Second Edition” (replacing \( S^+ \) by \( S \)).*
Q-learning versus Sarsa

Example 6.6 of “Reinforcement Learning: An Introduction, Second Edition”.

[Diagram showing the comparison between Q-learning and Sarsa, with a grid and cumulative rewards over episodes.]
Because behaviour policy in Q-learning is $\varepsilon$-greedy variant of the target policy, the same samples (up to $\varepsilon$-greedy) determine both the maximizing action and estimate its value.

Figure 6.5 of "Reinforcement Learning: An Introduction, Second Edition".
Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in S$, $a \in A(s)$, such that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
    Take action $A$, observe $R$, $S'$
    With 0.5 probability:
      \[ Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left( R + \gamma Q_2(S', \text{argmax}_a Q_1(S', a)) - Q_1(S, A) \right) \]
    else:
      \[ Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left( R + \gamma Q_1(S', \text{argmax}_a Q_2(S', a)) - Q_2(S, A) \right) \]
    $S \leftarrow S'$
  until $S$ is terminal

So far, all methods were **on-policy**. The same policy was used both for generating episodes and as a target of value function.

However, while the policy for generating episodes needs to be more exploratory, the target policy should capture optimal behaviour.

Generally, we can consider two policies:

- **behaviour** policy, usually $b$, is used to generate behaviour and can be more exploratory;
- **target** policy, usually $\pi$, is the policy being learned (ideally the optimal one).

When the behaviour and target policies differ, we talk about **off-policy** learning.
On-policy and Off-policy Methods

The off-policy methods are usually more complicated and slower to converge, but are able to process data generated by different policy than the target one.

The advantages are:

- can compute optimal non-stochastic (non-exploratory) policies;
- more exploratory behaviour;
- ability to process expert trajectories.
Off-policy Prediction

Consider prediction problem for off-policy case.

In order to use episodes from $b$ to estimate values for $\pi$, we require that every action taken by $\pi$ is also taken by $b$, i.e.,

$$\pi(a|s) > 0 \Rightarrow b(a|s) > 0.$$  

Many off-policy methods utilize **importance sampling**, a general technique for estimating expected values of one distribution given samples from another distribution.
Assume that $b$ and $\pi$ are two distributions and let $x_i$ be $N$ samples of $b$. We can then estimate $\mathbb{E}_{x \sim b}[f(x)]$ as

$$\mathbb{E}_{x \sim b}[f(x)] \sim \frac{1}{N} \sum_i f(x_i).$$

In order to estimate $\mathbb{E}_{x \sim \pi}[f(x)]$ using the samples $x_i$, we need to account for different probabilities of $x_i$ under the two distributions by

$$\mathbb{E}_{x \sim \pi}[f(x)] \sim \frac{1}{N} \sum_i \frac{\pi(x_i)}{b(x_i)} f(x_i)$$

with $\pi(x)/b(x)$ being a relative probability of $x$ under the two distributions.
Off-policy Prediction

Given an initial state $S_t$ and an episode $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$, the probability of this episode under a policy $\pi$ is

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

Therefore, the relative probability of a trajectory under the target and behaviour policies is

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

Therefore, if $G_t$ is a return of episode generated according to $b$, we can estimate

$$
v_\pi(S_t) = \mathbb{E}_b [\rho_t G_t].
$$
Off-policy Monte Carlo Prediction

Let $\mathcal{T}(s)$ be a set of times when we visited state $s$. Given episodes sampled according to $b$, we can estimate

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

Such simple average is called **ordinary importance sampling**. It is unbiased, but can have very high variance.

An alternative is **weighted importance sampling**, where we compute weighted average as

$$v_\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_t G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t}.$$  

Weighted importance sampling is biased (with bias asymptotically converging to zero), but has smaller variance.
As a simple example, consider the 10-armed bandits from the first lecture, with single-step episodes.

Let the \textit{behaviour policy} be a uniform policy, so the return is a reward of a randomly selected arm.

Let the \textit{target policy} be a greedy policy always using action 3.

Assume that the first sample from the behaviour policy produced action 3 with reward $R$. Then

- Ordinary importance sampling estimate the return for the target policy as

$$\frac{\pi(a)}{b(a)} R = \frac{1}{1/10} R = 10 \cdot R.$$  

The factor 10 is present, because the behaviour policy returns action 3 in 10\% cases.

- Weighted importance sampling estimate the return for target policy as average of rewards for action 3.
Comparison of ordinary and weighted importance sampling on Blackjack. Given a state with sum of player's cards 13 and a usable ace, we estimate target policy of sticking only with a sum of 20 and 21, using uniform behaviour policy.
Off-policy Monte Carlo Policy Evaluation

We can compute weighted importance sampling similarly to the incremental implementation of Monte Carlo averaging.

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

- **Input:** an arbitrary target policy $\pi$
- **Initialize, for all $s \in S$, $a \in A(s)$:**
  - $Q(s, a) \in \mathbb{R}$ (arbitrarily)
  - $C(s, a) \leftarrow 0$
- **Loop forever (for each episode):**
  - $b \leftarrow$ any policy with coverage of $\pi$
  - 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 $W \neq 0$:**
    - $G \leftarrow \gamma G + R_{t+1}$
    - $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    - $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

*Algorithm 5.6 of "Reinforcement Learning: An Introduction, Second Edition".*
Off-policy Monte Carlo

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

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

- $Q(s, a) \in \mathbb{R}$ (arbitrarily)
- $C(s, a) \leftarrow 0$
- $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$ (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, $t = T-1, T-2, \ldots, 0$:

- $G \leftarrow \gamma G + R_{t+1}$
- $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
- $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$ (with ties broken consistently)
- If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
- $W \leftarrow W b(A_t|S_t)$

Algorithm 5.7 of "Reinforcement Learning: An Introduction, Second Edition".
The action $A_{t+1}$ is a source of variance, providing correct estimate only in expectation.

We could improve the algorithm by considering all actions proportionally to their policy probability, obtaining Expected Sarsa algorithm:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}_\pi q(S_{t+1}, a) - q(S_t, A_t) \right]$$

$$\leftarrow q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum_a \pi(a | S_{t+1}) q(S_{t+1}, a) - q(S_t, A_t) \right].$$

Compared to Sarsa, the expectation removes a source of variance and therefore usually performs better. However, the complexity of the algorithm increases and becomes dependent on number of actions $|\mathcal{A}|$. 
Expected Sarsa as an Off-policy Algorithm

Note that Expected Sarsa is also an off-policy algorithm, allowing the behaviour policy $b$ and target policy $\pi$ to differ.

Especially, if $\pi$ is a greedy policy with respect to current value function, Expected Sarsa simplifies to Q-learning.
Asymptotic performance is averaged over 100k episodes, interim performance over the first 100.

Figure 6.3 of "Reinforcement Learning: An Introduction, Second Edition".
$n$-step Methods

Full return is

\[ G_t = \sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1}, \]

one-step return is

\[ G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1}). \]

We can generalize both into $n$-step returns:

\[ G_{t:t+n} \overset{\text{def}}{=} \left( \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} \right) + \gamma^n V(S_{t+n}). \]

with $G_{t:t+n} \overset{\text{def}}{=} G_t$ if $t + n \geq T$ (episode length).
A natural update rule is

\[ V(S_t) \leftarrow V(S_t) + \alpha \left[ G_{t:t+n} - V(S_t) \right]. \]

**n-step TD for estimating \( V \approx v_\pi \)**

- **Input:** a policy \( \pi \)
- **Algorithm parameters:** step size \( \alpha \in (0, 1] \), a positive integer \( n \)
- **Initialize** \( V(s) \) arbitrarily, for all \( s \in S \)
- **All store and access operations** (for \( S_t \) and \( R_t \)) can take their index mod \( n + 1 \)

**Loop for each episode:**

Initialize and store \( S_0 \neq \text{terminal} \)

\( T \leftarrow \infty \)

Loop 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 \( S_{t+1} \) 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 \( \tau = T - 1 \)

*Algorithm 7.1 of "Reinforcement Learning: An Introduction, Second Edition".*
Using the random walk example, but with 19 states instead of 5,

we obtain the following comparison of different values of $n$: 

![Graph showing comparison of different values of $n$.](image)

*Figure 7.2 of "Reinforcement Learning: An Introduction, Second Edition".*
Defining the $n$-step return to utilize action-value function as

$$G_{t:t+n} \overset{\text{def}}{=} \left( \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} \right) + \gamma^n Q(S_{t+n}, A_{t+n})$$

with $G_{t:t+n} \overset{\text{def}}{=} G_t$ if $t + n \geq T$, we get the following straightforward algorithm:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ G_{t:t+n} - Q(S_t, A_t) \right].$$
$n$-step Sarsa Algorithm

Initialize $Q(s, a)$ arbitrarily, for all $s \in S, a \in A$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$, or to a fixed given policy
Algorithm 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 $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
          $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
        If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$

Algorithm 7.2 of "Reinforcement Learning: An Introduction, Second Edition".
Off-policy $n$-step Sarsa

Recall the relative probability of a trajectory under the target and behaviour policies, which we now generalize as

$$
\rho_{t:t+n} \overset{\text{def}}{=} \prod_{k=t}^{\min(t+n, T-1)} \frac{\pi(A_k | S_k)}{b(A_k | S_k)}.
$$

Then a simple off-policy $n$-step TD policy evaluation can be computed as

$$
V(S_t) \leftarrow V(S_t) + \alpha \rho_{t:t+n-1} [G_{t:t+n} - V(S_t)].
$$

Similarly, $n$-step Sarsa becomes

$$
Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \rho_{t+1:t+n} [G_{t:t+n} - Q(S_t, A_t)].
$$
Off-policy $n$-step Sarsa

Off-policy $n$-step Sarsa for estimating $Q \approx q_\pi$ or $q_\pi$

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 greedy with respect to $Q$, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer $n$
All store and access operations (for $S_t, A_t$, and $R_t$) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop 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 $A_{t+1} \sim b(\cdot|S_{t+1})$
        $\tau \leftarrow t - n + 1$  (where is the time whose estimate is being updated)
        If $\tau \geq 0$:
          $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n,T-1)} \pi(A_i|S_i)$
          $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
          If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
          $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \rho [G - Q(S_t, A_t)]$
        If $\pi$ is being learned, then ensure that $\pi(\cdot|S_t)$ is greedy wrt $Q$
    Until $\tau = T - 1$

Modified from Algorithm 7.3 of "Reinforcement Learning: An Introduction, Second Edition" by changing $\rho_{\{i+1:i+n-1\}}$ to $\rho_{\{i+1:i+n\}}$. 
Q-learning and Expected Sarsa can learn off-policy without importance sampling.

To generalize to $n$-step off-policy method, we must compute expectations over actions in each step of $n$-step update. However, we have not obtained a return for the non-sampled actions.

Luckily, we can estimate their values by using the current action-value function.
We now derive the \(n\)-step reward, starting from one-step:

\[
G_{t:t+1} \overset{\text{def}}{=} R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a).
\]

For two-step, we get:

\[
G_{t:t+2} \overset{\text{def}}{=} R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+2}.
\]

Therefore, we can generalize to:

\[
G_{t:t+n} \overset{\text{def}}{=} R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1})Q(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1})G_{t+1:t+n}.
\]

The resulting algorithm is \(n\)-step Tree backup and it is an off-policy \(n\)-step temporal difference method not requiring importance sampling.
**Off-policy $n$-step Without Importance Sampling**

### $n$-step Tree Backup for estimating $Q \approx q_*$ or $q_*$

Initialize $Q(s,a)$ arbitrarily, for all $s \in S, a \in A$

Initialize $\pi$ to be greedy with respect to $Q$, or as a fixed given policy

Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer $n$

All store and access operations can take their index mod $n + 1$

Loop for each episode:

- Initialize and store $S_0 \neq$ terminal
- Choose an action $A_0$ arbitrarily as a function of $S_0$; Store $A_0$
- $T \leftarrow \infty$
- Loop for $t = 0, 1, 2, \ldots$
  - If $t < T$:
    - Take action $A_t$; observe and store the next reward and state as $R_{t+1}, S_{t+1}$
    - If $S_{t+1}$ is terminal:
      - $T \leftarrow t + 1$
    - else:
      - Choose an action $A_{t+1}$ arbitrarily as a function of $S_{t+1}$; Store $A_{t+1}$
  - $\tau \leftarrow t + 1 - n$ (\(\tau\) is the time whose estimate is being updated)
- If $\tau \geq 0$:
  - If $t + 1 \geq T$:
    - $G \leftarrow R_T$
  - else:
    - $G \leftarrow R_{t+1} + \gamma \sum_{a} \pi(a|S_{t+1})Q(S_{t+1}, a)$
    - Loop for $k = \min(t, T - 1)$ down through $\tau + 1$:
      - $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k)Q(S_k, a) + \gamma \pi(A_k|S_k)G$
      - $Q(S_{\tau}, A_{\tau}) \leftarrow Q(S_{\tau}, A_{\tau}) + \alpha [G - Q(S_{{\tau}}, A_{{\tau}})]$
  - If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt $Q$

Until $\tau = T - 1$

---

*Algorithm 7.5 of "Reinforcement Learning: An Introduction, Second Edition".*

NPFL122, Lecture 3  Refresh Afterstates TD Q-learning Double Q Off-policy Expected Sarsa $n$-step TB 42/42