1 n-step Bootstrapping

- unify the MC and one-step tabular TD methods
- n-step TD methods generalize both by spanning a spectrum with MC on one end, and one-step TD at the other
- n-step methods enable bootstrapping over multiple time steps

1.1 n-step TD Prediction

Consider estimate v_π from sample episodes generated using π

- MC methods perform an update for each visited state based on the entire sequence of rewards from that state until the end of the episode
- one-step TD methods perform an update based on the next reward only
- n-step: in-between

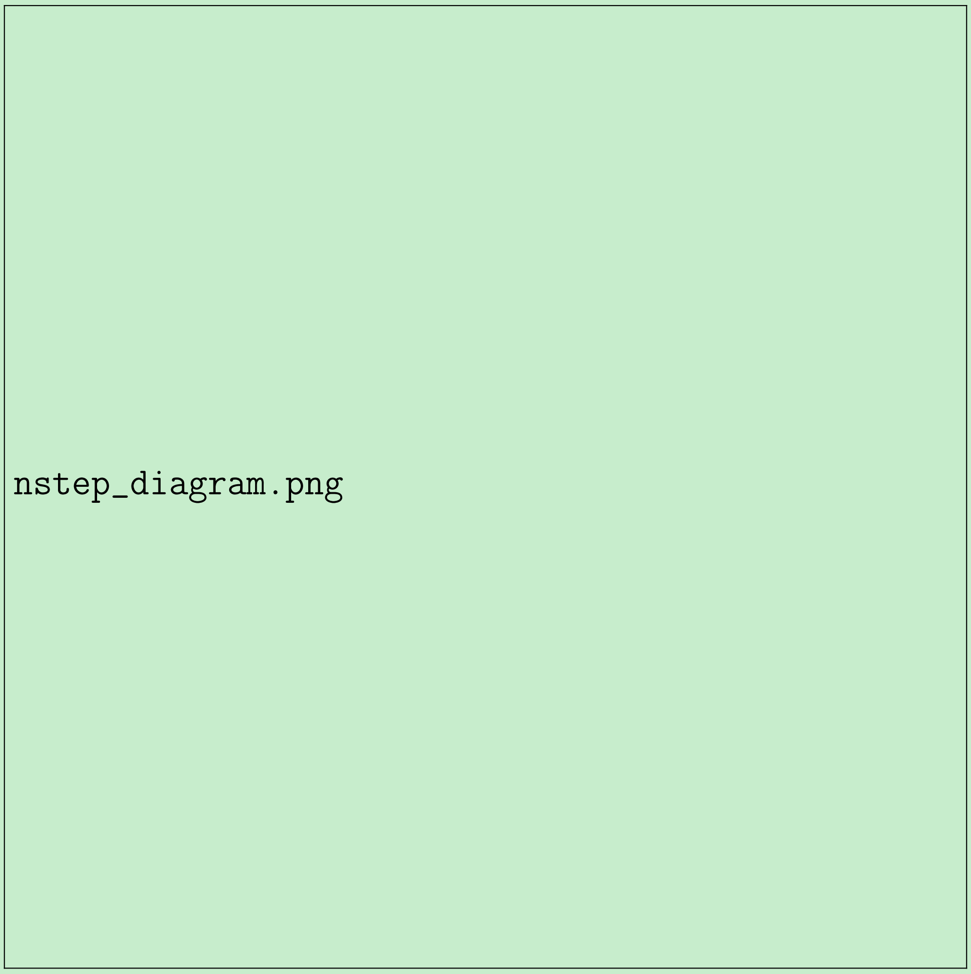
Still TD: we bootstrap because we change an earlier estimate based on how it differs from a later estimate (only that the later estimate is now n steps later)

1.1.1 Targets

notation: $G_t^{(n)} = G_{t:t+n}$, the return from time step t to $t+n$

$$\begin{aligned}
n &= 1 \quad (TD(0)) & G_t^{(1)} &= R_{t+1} + \gamma V_t(S_{t+1}) && (7.1)\\
n &= 2 & G_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) && ()\\
&\vdots & &\vdots && ()\\
n &= \infty \quad (MC) & G_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T && (7.2)
\end{aligned}$$

- In MC, the target is the return
- In one-step TD, the target is the one-step return, that is the first reward plus the discounted estimated value of the next state
- $V_t : \mathcal{S} \rightarrow \mathbb{R}$ is the estimate at time t of v_π.



(a)

Figure 1: Backup diagrams for n-step methods

n-step target:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{7.3}$$

- All n-step returns can be considered approximations of the full return, truncated after n steps and then corrected for the remaining steps by $V_{t+n-1}(S_{t+n})$
- n-steps return for $n \geq 1$ involve future rewards that are not available from the transition $t \rightarrow t+1$; we will see eligibility traces for an online method not involving future rewards
- plugging this into the state-value learning

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha[G_t^{(n)} - V_{t+n-1}(S_t)] \tag{7.4}$$

- no changes are made during the first (n-1) steps of each episode. To make up for that, an equal number of updates are made after the termination of the episode, before starting the next.

(a)

Figure 2: n-step TD algorithm

1.1.2 Error-reduction Property

The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}. An important property of the n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is in a worst-state sense. The worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V_{t+n-1}.

$$\max_s |\mathbb{E}_\pi[G_t^{(n)}|S_t = s] - v_\pi(s)| \leq \gamma^n \max_s |V_{t+n-1}(s) - v_\pi(s)| \tag{7.5}$$

This is called the **error reduction property**. We can use this to show that all n-steps methods converge to the correct predictions under appropriate technical conditions.

We use this to type of graph to show the effect of n

`nstep_graph.png`

(a)

Figure 3: Performance of n-step TD methods as a function of α

1.2 n-step SARSA

- use n-step not just for prediction but also for control
- change from states to state-action pairs, and use a ε-greedy policy
- The backup diagrams for n-step SARSA are like those of n-step TD, except that the SARSA ones start and end in a state-action node. We redefine the n-step return in terms of estimated action-values instead of state-values

$$G_{t:t+n} = R_{t+1} + \gamma R_{T+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{7.6}$$

- and plug that into our GPI update

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha\big(G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\big) \tag{7.7}$$

The values of all other states remain unchanged, $Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$ for $s \neq S_t$ or $a \neq A_t$.

1.2.1 Expected SARSA

What about n-step version of expected SARSA? The backup diagram consists of a linear string of sample actions and states and the last element is a branch over all action possibilities, weighted by their probability of occurring under π. The n-step return is here defined by:

$$G_{t:t+n} = R_{t+1} + \gamma R_{T+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}, A_{t+n}) \tag{7.8}$$

Where $\bar{V}_t(s)$ is the expected approximate value of state s, using the estimated action values and the policy

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s, a)$$

- Expected approximate values are important!
- if s is terminal, then its approximated value is defined to be 0.

nstep_sarsa_algo.png

(a)

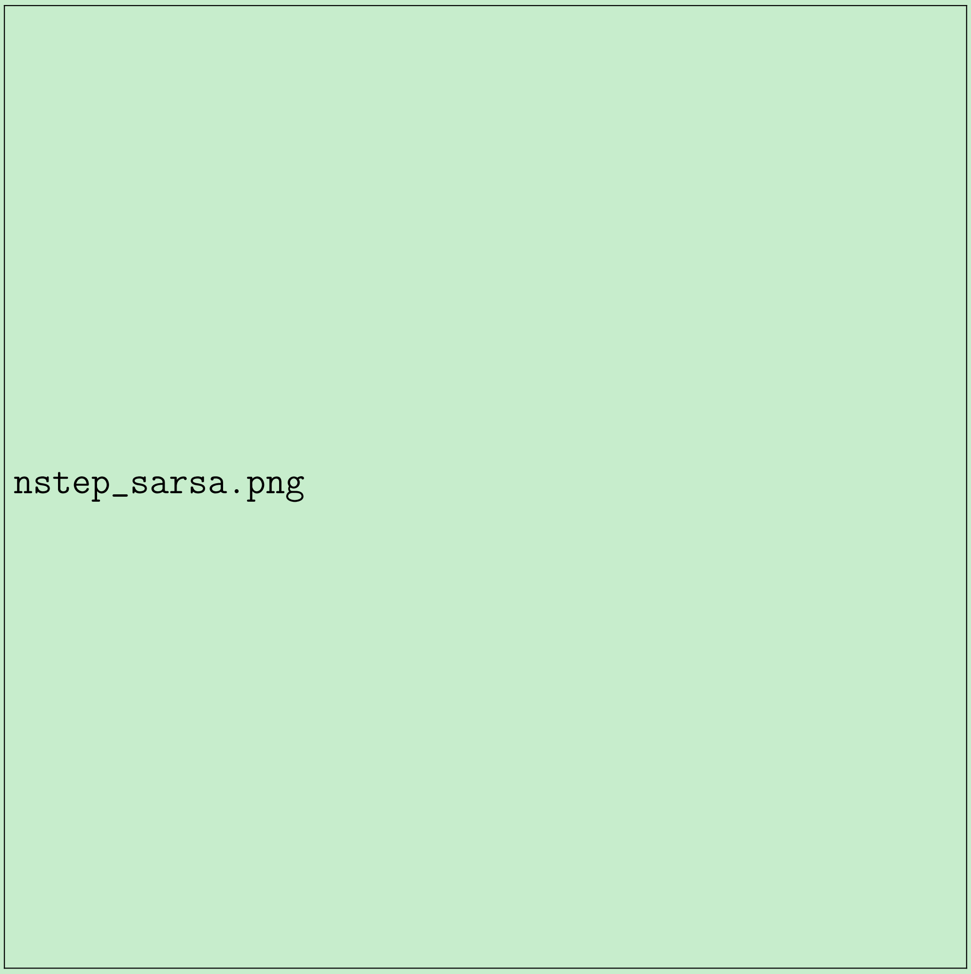
Figure 4: n-step SARSA

1.3 n-step Off-policy Learning (by importance sampling)

- To use the data from behavior policy b, we must take into account the relative probability of taking the actions that were taken
- In n-step methods, the returns are constructed over n steps so we are interested in the relative probability of just these n actions

For example, to make a simple version of n-step TD, we can weight the update for time t (made at time $t+n$) by the importance sampling ratio

$$\rho_{t:h} = \prod_{k=t}^{min(h,T-1)} \frac{\pi(A_k \| S_k)}{b(A_k \| S_k)} \tag{7.10}$$



(a)

Figure 5: n-step SARSA backup diagram

If we plug this into our update, we get

$$V_{t+n}(S_t) \leftarrow V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}(G_{t:t+n} - V_{t+n-1}(S_t))$$

Similarly, the n-step SARSA update can be extended for the off-policy method

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n-1}(G_{t:t+n} - Q_{t+n-1}(S_t, A_t))$$

Note that the IS ratio here starts one step later than from n-step TD because we are updating a state-action pair (we know that we have selected the action so we need IS only for the subsequent actions).

The off-policy version for **expected SARSA** would be the same except that the IS ratio would use $\rho_{t+1:t+n-2}$ instead of $\rho_{t+1:t+n-1}$ because all possible nstep_sarsa_offpolicy.png

(a)

Figure 6: n-step SARSA off-policy algorithm

actions are taken into account *in the last state*, the one actually taken has no effect and does not need to be corrected for.

why Q-learning can be off-policy w/o IS?

Q-learning bootstraps off of the largest action value in its next state. This is like sampling an action under an estimate of the optimal policy rather than the behavior policy. Q-learning learns about the best action it could take rather the actions it actually takes, so it's off-policy.

why expected SARSA can be off-policy w/o IS?

the expectation over actions is computed independently of the action actually selected in the next state, so it's off-policy

why Q-learning doesn't use IS?

https://stats.stackexchange.com/questions/335396/why-dont-we-use-impo
For action values, the fist action does not play a role in the IS ratio (it has been taken!).

1.4 Per-decision Methods with Control Variates

The multi-step off-policy methods presented in 7.3 is conceptually clear but not very efficient. What about per-decision sampling (recall Chap 5.9).

1. The n-step return can be written recursively. For the n steps being ending at horizon h, the n-step return can be written $G_{t:h} = R_{t+1} + \gamma R_{t+1:h}$

2. Consider the effect of following behavior policy b is not the same as following policy π, so all the resulting experiences, including the first reward R_{t+1} and the next state S_{t+1}, must be weight by the importance sampling ratio for time t, $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

A simple approach would be to weight the right-hand side $R_{t+1} + \gamma G_{t+1:h}$ by ρ_t. A more sophisticated approach would be to use an off-policy definition of the n-step return ending in horizon h

$$G_{t:h} \doteq \rho_t(R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V_{h-1}(S_t) \tag{7.11}$$

where $t < h < T$, and $G_{h:h} \doteq V_{h-1}(S_h)$.

If $\rho_t = 0$ (trajectory has no chance to occur under the target policy), then instead of changing the target $G_{t:h}$ to be 0 and causing the estimate to shrink, the target is the same as the estimate and cause no change. The additional term $(1 - \rho_t)V_{h-1}(S_t)$ is called a **control variate** and conceptually is here to ensure the idea that if $\rho_t =$, then we should ignore the sample and don't change the estimate.

We can otherwise use the conventional learning rule that does not have explicit IS ratios, except the one in $G_{t:h}$.

1.4.1 For action values

For action values, the off-policy definition of the n-step return is a little different because the fist action does not play a role in the IS ratio (it has been taken!).

The n-step on-policy return ending at horizon h can be written recursively, and for action-values, the recursion ends with $G_{h:h} \doteq \bar{V}_{h-1}(S_h)$. An off-policy form with control variate is:

$$G_{t:h} \doteq R_{t+1} + \gamma\big(\rho_{t+1}G_{t+1:h} + \bar{V}_{h-1}(S_{t+1}) - \rho_{t+1}Q_{h-1}(S_{t+1}, A_{t+1})\big) \quad (7.12)$$
$$= R_{t+1} + \gamma\rho_{t+1}\big(G_{t+1:h}Q_{h-1}(S_{t+1}, A_{t+1})\big) + \gamma\bar{V}_{h-1}(S_{t+1})\big) \quad (7.13)$$

for $t < h < T$.

- if $h < T$, then the recursion end with $G_{h:h} \doteq (S_h, A_h)$
- if $h = T$, it ends with $G_{T-1:T} \doteq R_T$. The resultant prediction algorithm is analogous to Expected SARSA.

Off-policy methods have higher variance and it's probably inevitable. Things to help reduce the variance:

- control variates
- adapt the step size parameter to the observed variance
- invariant updates

1.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

Off-policy without IS: Q-learning and Expected SARSA do it for the one-step case. Here, we present a multi-step algorithm: **the tree backup algorithm**.

A 3-step tree backup diagram

- 3 sample states and rewards
- 2 sample actions

nstep_tree.png

(a)

Figure 7: n-step tree

- list of unselected actions for each state
- we have no sample for the unselected actions, and we bootstrap with their estimated values

So far, the target for the update of a node was combining the rewards along the way and the estimated values of the nodes at the bottom. Now we add to this the estimated values of the actions not selected. This is why it is called a tree backup update: it is an update from the entire tree of estimated action values.

Each leaf node contributes to the target with a weight proportional to their probability of occurring under π.

- each first-level action a contributes with a weight of $\pi(a|S_{t+1})$
- the action actually taken, A_{t+1}, does not contribute and its probability

$\pi(A_{t+1}|S_{t+1})$ is used to weight all the second level action values

- each second-level action a' thus has a weight of $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$

The one-step return (target) is the same as Expected SARSA (for $t < T - 1$):

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a) \tag{7.14}$$

And the two-step tree-backup return is (for $t < T - 2$):

$$\begin{aligned} G_{t:t+2} \doteq R_{t+1} &+ \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_t(S_{t+1}, a) \\ &+ \gamma \pi(A_{t+1}|S_{t+1}) \big(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \big) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_t(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2} \end{aligned} \tag{1}$$

The latter form suggests the recursive form of the n-step tree backup return (for $t < T - 1$):

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \tag{7.15}$$

We then use this target in the usual action-value update from n-step SARSA:

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha \big(G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \big) \tag{7.16}$$

nstep_tree_backup.png

(a)

Figure 8: n-step tree backup algorithm

1.6 A Unifying Algorithm: n-step Q(σ)