7.0: Intro

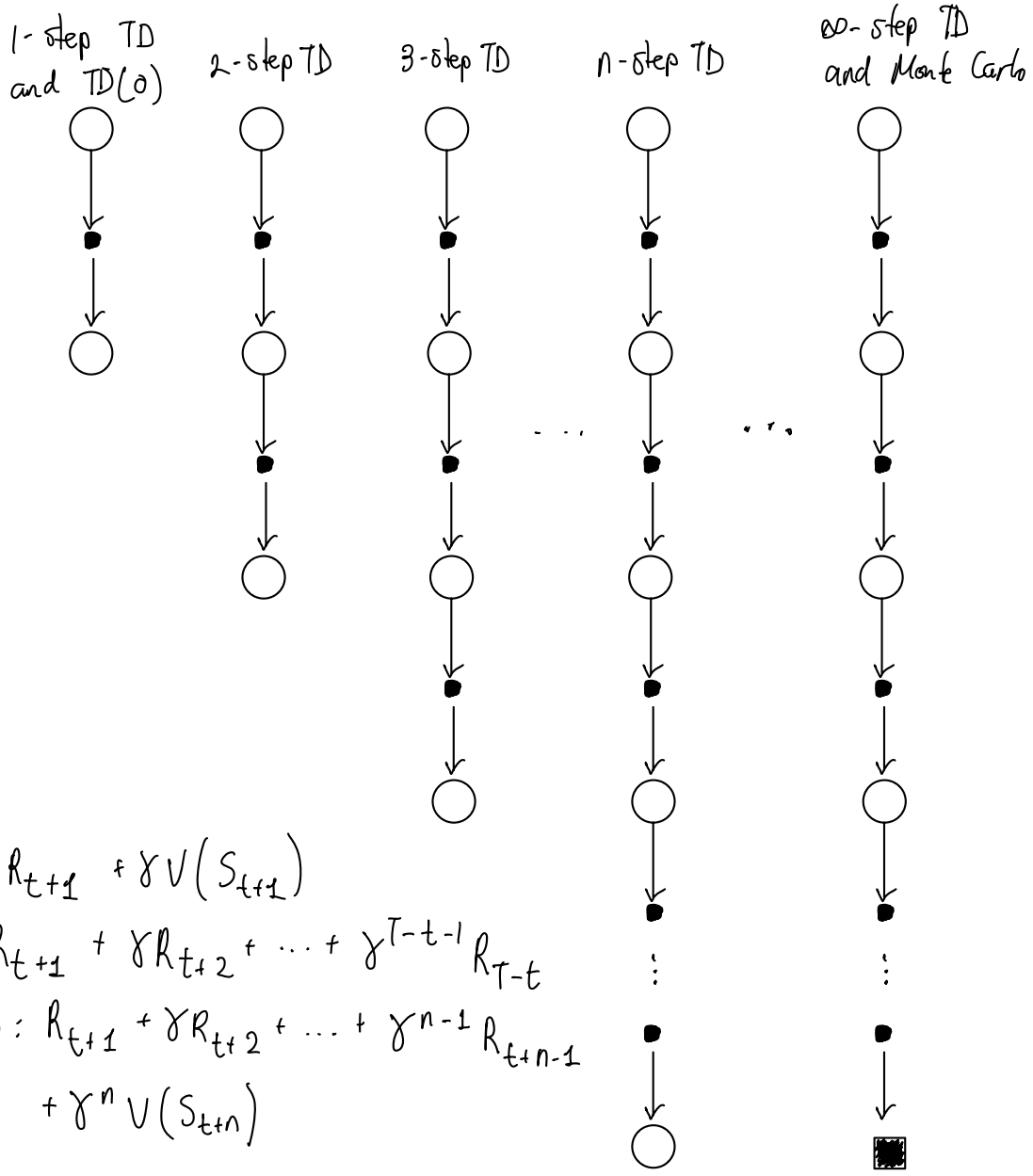
- n-step bootstrapping unifies the Monte Carlo (MC) and the one-step Temporal Difference (TD) methods.
  - TD is a combination of DP and MC.

- 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, which is nice because using only one time-step can reduce the power of our algorithms.

7.1: n-step Prediction

Consider estimating V_π from sample episodes generated using π.

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



Targets:

TD(0): $R_{t+1} + \gamma V(S_{t+1})$

MC: $R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_{T-t}$

n-step TD: $R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n-1} + \gamma^n V(S_{t+n})$

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 step later).

Targets

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

$n = 1$ (TD) $\quad G_t^{(1)} = R_{t+1} + \gamma V_t(S_{t+1})$

$n = 2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$

$\vdots \qquad \vdots$

$n = \infty$ (MC) $\quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$

- In MC: the target is the actual 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.
- $V_t : \mathcal{S} \to \mathbb{R}$ is the estimate at time t of v_π.

n-step return:

$$G_{t:t+n} \doteq \overbrace{R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n}}^{\text{actual rewards}} + \overbrace{\gamma^n V_{t+n-1}(S_{t+n})}^{\text{bootstrap!}}$$

- 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 $n > 1$ involve future rewards that are not available from the transition $t \to t+1$. We'll see eligibility traces (ch 12) for an online method not involving future rewards.

The update rule is:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \quad 0 \le t < T$$

- No changes are made during the first $n-1$ steps of each episodes to make up for that, an equal number of updates are made after the termination of the episode, before starting the next.

Algorithm: on-policy model-free prediction, Estimates state-value functions for a given policy π.

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

Input: a policy π // needs policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n // n = # of steps to look ahead.
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

Loop for each episode: // iterate thru episodes
  Initialize and store $S_0 \neq$ terminal // start at non-terminal state.
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$: // loop for time step
  | If $t < T$, then:
  | | Take an action according to $\pi(\cdot|S_t)$ // pick action according to policy π
  | | 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 $T \leftarrow t+1$ // record terminal time
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$: // ensures we perform update after at least n-steps
  | | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$ // sums the discounted rewards from $\tau+1$ to n steps ahead
  | | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ // update if eligible ($G_{\tau:\tau+n}$)
  | | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\left[G - V(S_\tau)\right]$ // value update
  Until $\tau = T - 1$

Error-Reduction Property

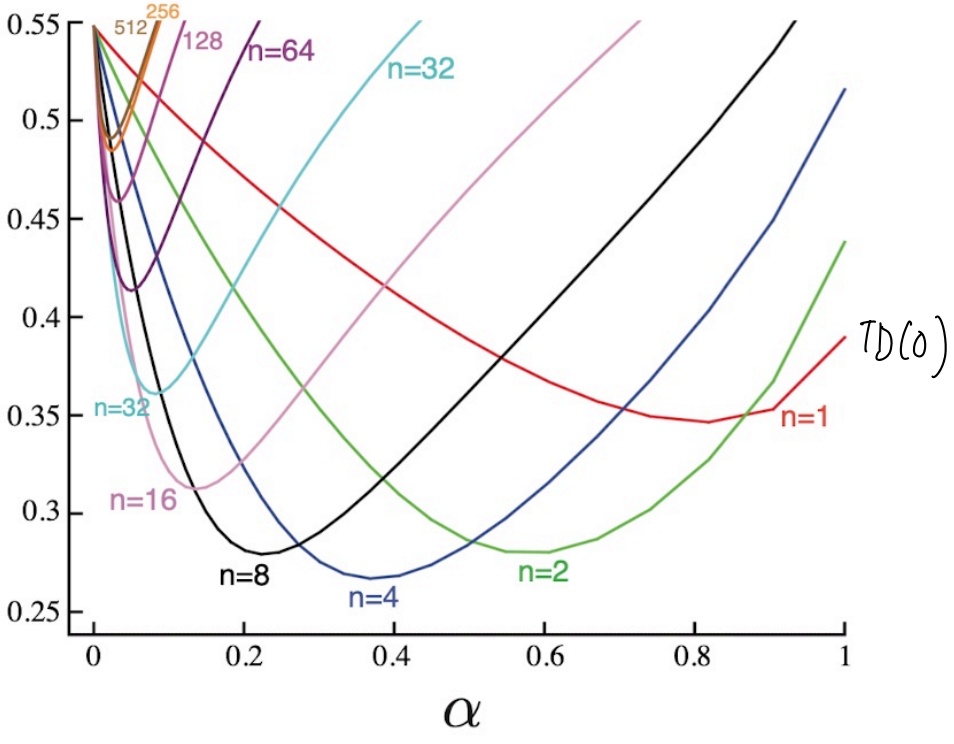
The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}. An important property of the n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense. The worst error of the expected n-step return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

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

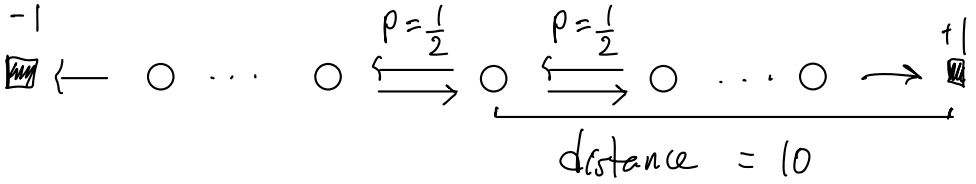
This is called the error reduction property. We can use this to show that all n-steps methods converge to the correct predictions under appropriate technical conditions.

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

Average RMS error over 19 states and first 10 episodes



Chain Markov Reward Process (length = 19):



Influence on state A value via the +1 reward

MC: must get to +1 goal at least one after visting A.

TD(0): must visit +1 goal at least once anytime and then wait for value to propagate.

Statistical Properties:

MC: unbiased, high variance.
Fast propagation of reward (all visited states).

TD(0): Biased updates, low variance.
Slow propagation of reward (most recent state).

n-step TD: medium variance
medium propagation of reward.

correct balance problem specific!

7.2: n-step SARSA (control)

- This section uses n-step for control.
- Uses state-action pairs and a ε-greedy policy, rather than state-values.
- The back-up diagrams for n-step SARSA are like those of n-step TD, except that the SARSA ones start and end in a state-action mode.
- We redefine the n-step return in terms of estimated action-values instead of state-values:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}),$$
$$n \geq 1, \quad 0 \leq t < T-n, \quad G_{t:t+n} = G_t \text{ if } t+n \geq T.$$

The GPI update rule is thus:

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha\left(G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right),$$
$$0 \leq t \leq T.$$

The values of all other states remain unchanged,

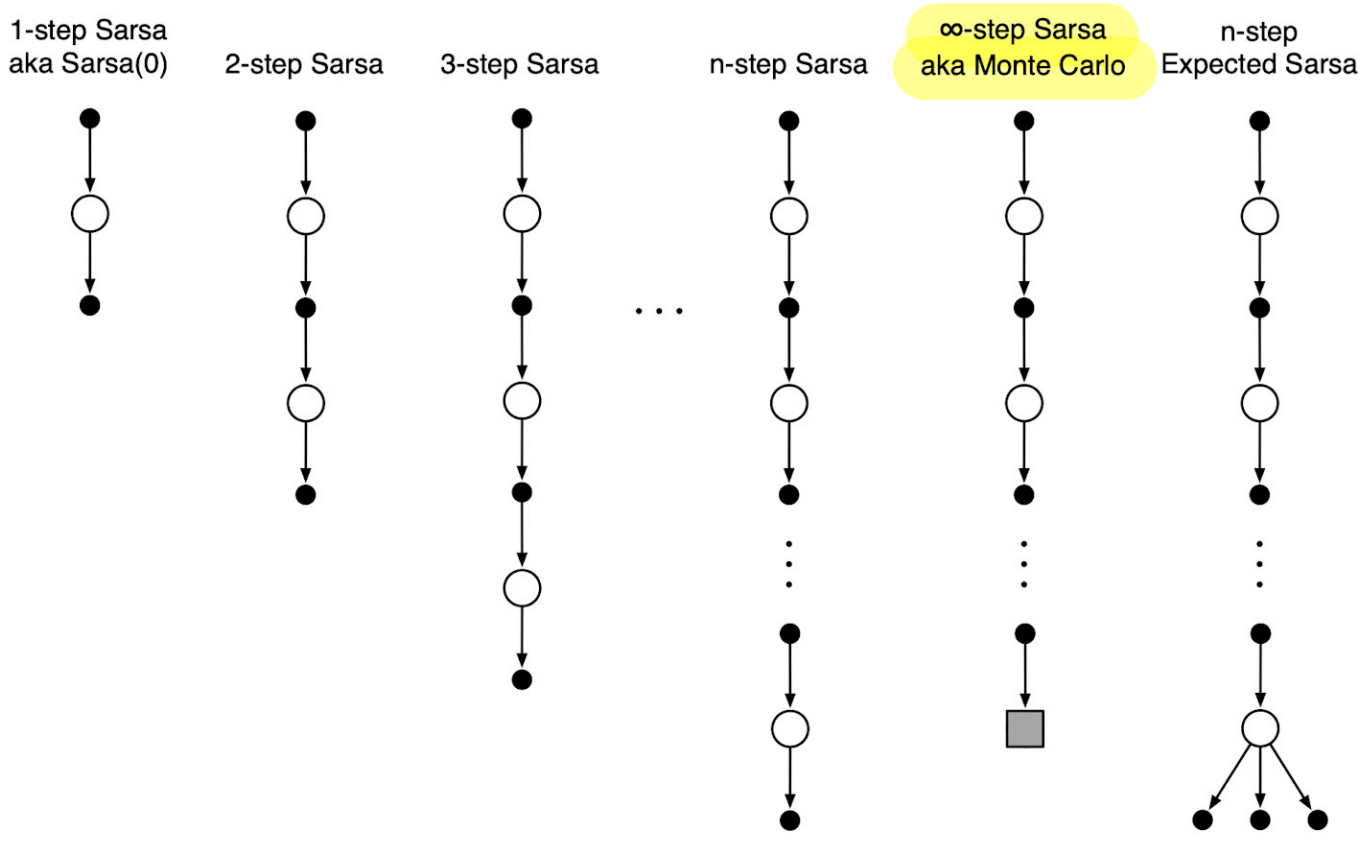
$$Q_{t+n}(s,a) = Q_{t+n-1}(s,a) \text{ for } s \neq S_t \text{ or } a \neq A_t.$$

Algorithm: on-policy control

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$ //arbitrary
Initialize π to be ε-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 $A_0 \sim \pi(\cdot|S_0)$ //use policy π to select Action A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | | Take action A_t //action A_t was selected at time $t-1$
  | | 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: //After arriving at next state S_{t+1}, use policy to select next action A_{t+1}
  | | | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$: //start updating after n-steps.
  | | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$ //actually selected for $S_{\tau+n}$
  | | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
  | | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q //policy is dynamic!
  Until $\tau = T - 1$



Expected SARSA:

What about n-step version of expected SARSA? The back-up 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:

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

for $t+n < T$, and 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.

7.3: n-step Off-Policy learning by importance sampling

- In order to use the data from the 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.

Off-policy version of n-step TD: weight the update for time t (made at time $t+n$) by the importance-sampling ratio:

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

// relative probability under the two policies of taking the n actions from A_t to A_{t+n-1}

Then the update is:

$$V_{t+n}(S_t) \leftarrow V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}\left(G_{t:t+n} - V_{t+n-1}(S_t)\right),$$
$$0 \leq t < T.$$

Note that if the two policies are actually the same (on-policy), the importance sampling ratio is always 1 and we have our previous update.

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

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

// off-policy version of n-step SARSA, by adding $\rho_{t:t+n-1}$

Note that the importance sampling 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 importance sampling only for the subsequent actions).

Algorithm: off policy version of n-step SARSA.

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

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$ } 2 distinct policies
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π 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 $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:\tau+n})$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
  | 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 π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

The off-policy version for expected SARSA would be the same except that the importance sampling ratio would use $\rho_{t+1:t+n-2}$ instead of $\rho_{t+1:t+n-1}$ because all possible actions are taken into account in the last state, the one actually taken has no effect and does not need to be corrected for.

7.4: Per-decision Off-Policy Methods with Control Variates

The multi-step off-policy method presented in 7.3 is conceptually clear but not very efficient. What about per-decision sampling?

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

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 weighted 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-handed 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:

$$G_{t:h} \doteq \rho_t\left(R_{t+1} + \gamma G_{t+1:h}\right) + \underbrace{\left(1-\rho_t\right)V_{h-1}(S_t)}_{\text{control variate.}},$$

where:

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

$$E[\rho_t] = 1 \Rightarrow E[1-\rho_t] = 0$$
$$\Rightarrow E[(1-\rho_t)V_{h-1}(S_t)] = 0$$

If ρ_t is zero (trajectory has no chance to occur under the target policy), then instead of the target $G_{t:h}$ to be zero and causing the estimate to shrink, the target is the same as the estimate and causes 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 the importance sampling ratio is 0, 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 importance sampling ratios except the one in $G_{t:h}$.

for Action Values:

For action values the off-policy definition of the n-step return is a little different because the first action does not play a role in the importance sampling ratio (since 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 variates is:

$$G_{t:h} \doteq R_{t+1} + \gamma\left(\rho_{t+1} G_{t+1:h} + \bar{V}_{h-1}(S_{t+1}) - \rho_{t+1} Q_{h-1}(S_{t+1}, A_{t+1})\right)$$

$$= R_{t+1} + \gamma\left(\rho_{t+1}\left(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\right) + \gamma \bar{V}_{h-1}(S_{t+1})\right)$$

for $t < h \leq T$.

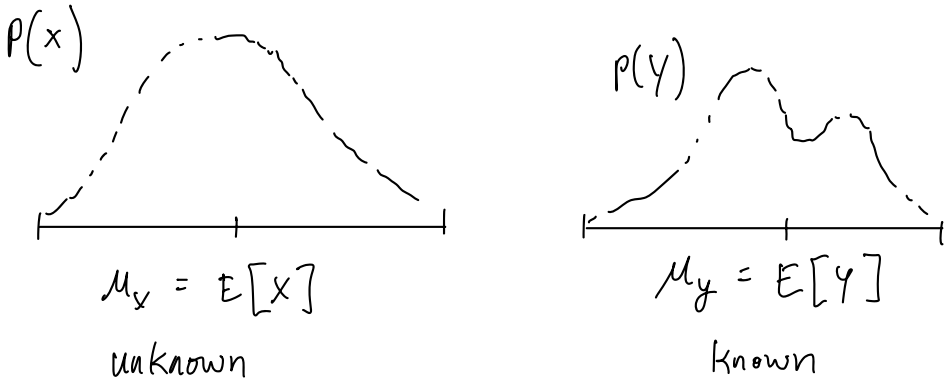
- If $h < T$: then the recursion ends with $G_{h:h} \doteq Q_{h-1}(S_h, A_h)$
- If $h = T$: it ends with $G_{T-1:T} \doteq R_T$. The resultant prediction algorithm is analogous to Expected SARSA.

Off-policy methods have higher variance.

Things to help reduce the variance:

- Control variates
- Adapt the step size to the observed variance.
- Invariant updates.

Control variates:



Assumption: X is positively correlated with Y

i.e. when $\mu_y > E[Y]$, it is likely that $\mu_x > E[X]$

ex: rain (X) and traffic (Y).

Sample $x \sim p(x)$ and $y \sim p(Y)$.

Consider two different estimators:

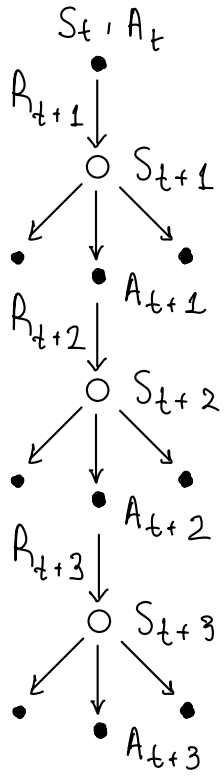
$\hat{\mu}_x = x$

$\hat{\mu}_x = X + \underbrace{[\mu_y - Y]}_{\text{Expectation}}$

0

7.5: Off-Policy without Importance Sampling: The n-step backup tree algorithm

Off-policy learning algorithm that performs update w/o importance sampling (Q-learning and Expected SARSA did it for the one-step case). Here, instead of following a single trajectory, it considers all possible actions of each step, weighting them by the target policy probabilities.



Example.

3-step tree backup diagram →

- 3 sample states and rewards (○)
- 2 sample actions
- list of unselected actions for each state
- we have no sample for the unselected actions, and we bootstrap with their estimated values.

How it works:

1) Tree structure: at each time-step, instead of sampling one action, it considers all possible actions weighted by the target policy $\pi(a|s)$.

2) Back-up Equations: The n-step tree back-up target is:
- starts with observed rewards along the actual path taken.
- At each level, backs up values for all actions weighted by target policy probabilities.

3) Update process:

- follows the actual behavioral policy path for rewards
- But backs up using expected values under the target policy
- creates a tree-like back-up diagram (hence the name).

So far, the target for the update of a state was combining the downstream rewards along the tranversed trajectory, plus the estimated values of the nodes at the end.

Now, we add to this the estimated values of the actions not selected. This is why it's called a tree back-up update: it is a weighted 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 weight of $\pi(a \mid S_{t+1})$
- The action actually taken, A_{t+1}, does not contribute and its probability $\pi(A_{t+1} \mid S_{t+1})$ is used to weight all the second level action values.
- Each second-level action a' thus has a weight of $\pi(A_{t+1} \mid S_{t+1})\, \pi(a' \mid S_{t+2})$.

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

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

↖ all actions (as in Expected SARSA).

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

$$G_{t:t+2} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_t(S_{t+1}, a) \quad \text{// actions not taken}$$

$$+ \gamma \pi(A_{t+1}|S_{t+1}) \left[R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \right] \quad \text{// action taken and downstream expected values.}$$

$$= \underbrace{R_{t+1}}_{\text{current reward}} + \underbrace{\gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_t(S_{t+1}, a)}_{\text{weighted returns of actions not taken}} + \underbrace{\gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2}}_{\text{weighted return of the taken action and its downstream weight returns. Note the recursion!}}$$

The n-step tree backup return (for $t < T-1$) is thus:

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \quad \text{// next-step non-taken actions}$$

$$+ \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n} \quad \text{// The next level of tree, using recursion.}$$

The update is thus:

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

Relationship to other methods:

- When $n = 1$: reduces to expected SARSA.
- When target policy = behavioral policy: becomes the on-policy n-step SARSA!

Algorithm: used to estimate q_*, off-policy.

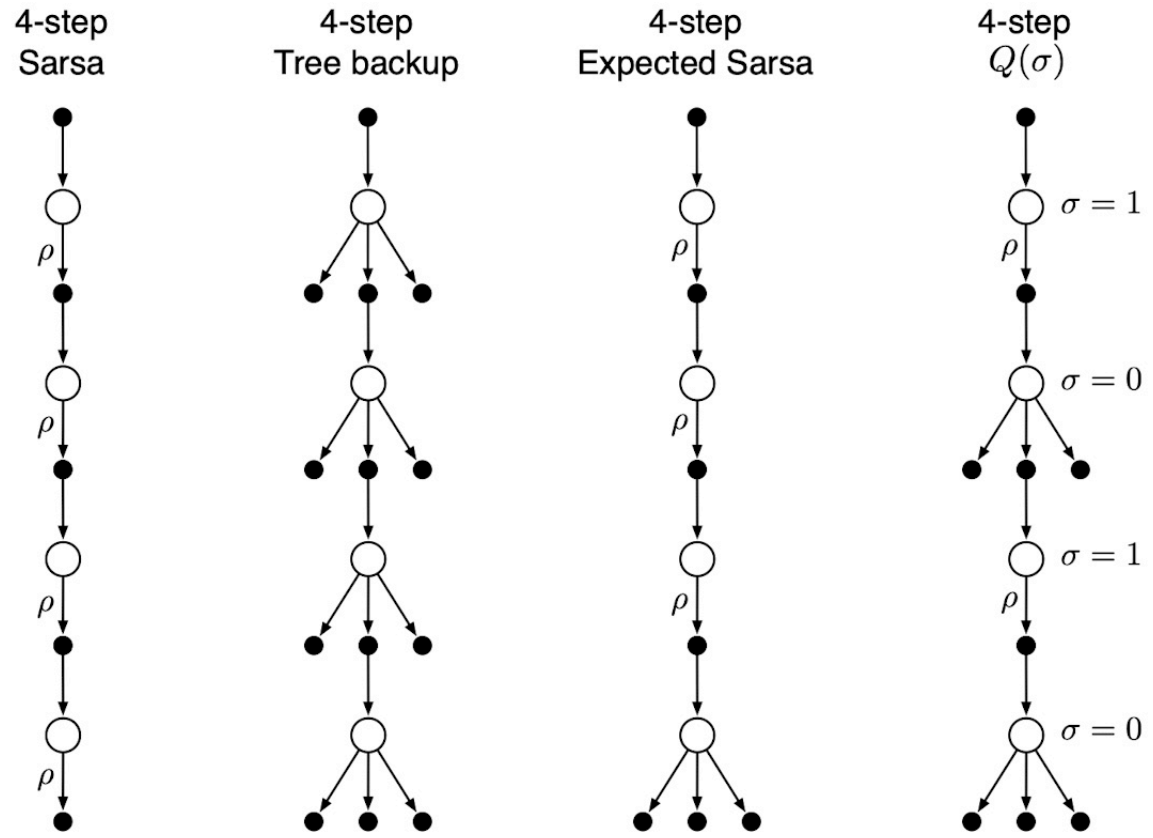
n-step Tree Backup for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy // greedy or fixed
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n = # look-ahead step.
All store and access operations can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal // initialize in non-terminal state.
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$: // for each time step in the episode.
  | If $t < T$: // checks if episode is still ongoing.
  | | 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}
  | $\tau \leftarrow t+1-n$ (τ is the time whose estimate is being updated) // update n-steps in the past.
  | If $\tau \geq 0$: // ensure we perform update after n-steps in episode.
  | | If $t+1 \geq T$: // if episode ended within the n-step lookahead window.
  | | | $G \leftarrow R_T$ // target is the final reward received.
  | | else // if episode is still ongoing:
  | | | $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$ // recursive process.
  | | Loop for $k = \min(t, T-1)$ down through $\tau+1$: // iterates from end to beginning of window.
  | | | $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
  | | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$ // Q-learning update.
  | | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

7.6: A Unifying algorithm: n-step $Q(\sigma)$

So far, we have considered 3 kinds of action-value algorithms, the 3 first of the next figure:



- n-step SARSA has all sample transitions,
- the tree backup has all state-to-action transitions branched without sampling.
- n-step Expected SARSA has all sample transitions except for the last state which is fully branched with an expected value.

Unification algorithm: the last diagram.

Basic idea: decide on a step-by-step basis whether we want to sample as in SARSA or consider the

expectation over all actions instead, as in the tree backup update. How?

- Let $\sigma_t \in [0, 1]$ denote the degree of sampling on step t, 1 being full sampling and 0 taking the expectation.
- The random variable σ_t might be set as a function of the state, or state-action pair, at time t.

This algorithm is called n-step Q(σ). To develop its equations, first we need to write the tree-backup n-step return in terms of the horizon $h = t + n$ and the expected approximate value $\bar{V}$:

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

$$= R_{t+1} + \gamma \bar{V}_{h-1}(S_{t+1}) - \gamma \pi(A_{t+1}|S_{t+1}) - Q_{h-1}(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:h}$$

$$= R_{t+1} + \gamma \pi(A_{t+1}|S_{t+1}) \left[G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}) \right] + \gamma \bar{V}_{h-1}(S_{t+1})$$

After which this is exactly like the n-step return for SARSA with control variates, except with the action probability $\pi(A_{t+1}|S_{t+1})$ substituted for the importance sampling ratio ρ_{t+1}. For Q(σ), we slidely linearly btwn these 2 cases:

$$G_{t:h} \doteq R_{t+1} + \gamma\Big[\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1})\Big]\cdot$$
$$\Big[G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\Big] + \gamma\bar{V}_{h-1}(S_{t+1})$$

for $t < h \leq T$. The recursion ends with $G_{h:h} \doteq 0$ if $h < T$ or with $G_{T-1:T} = R_T$ if $h = T$.

once this return is well-defined, we can plug it into the usual n-step SARSA update.

Off-policy n-step $Q(\sigma)$ for estimating $Q \approx q_*$ or q_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 π to be ε-greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, 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 //start in nonterminal state.
  Choose and store an action $A_0 \sim b(\cdot|S_0)$ //choose from policy b.
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$: //for every timestep.
  | If $t < T$: //At each time step:
  | 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 and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | Select and store σ_{t+1}
  | Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as ρ_{t+1} //calculate ρ for time-step.
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$: // if we are $\geq n$ steps in:
  | If $t+1 < T$: // if nonterminal:
  | $G \leftarrow Q(S_{t+1}, A_{t+1})$ //G will be our multi-step return; initialized to Q-value n steps ahead.
  | Loop for $k = \min(t+1, T)$ down through $\tau + 1$: //work backwards
  | if $k = T$: // terminal step
  | $G \leftarrow R_T$ //final reward
  | else: // nonterminal step
  | $\bar{V} \leftarrow \sum_a \pi(a|S_k) Q(S_k, a)$ //expected value of state under target policy π
  | //update return $G \leftarrow R_k + \gamma\big(\sigma_k \rho_k + (1-\sigma_k)\pi(A_k|S_k)\big)\big(G - Q(S_k, A_k)\big) + \gamma \bar{V}$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$ //update state-action value.
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

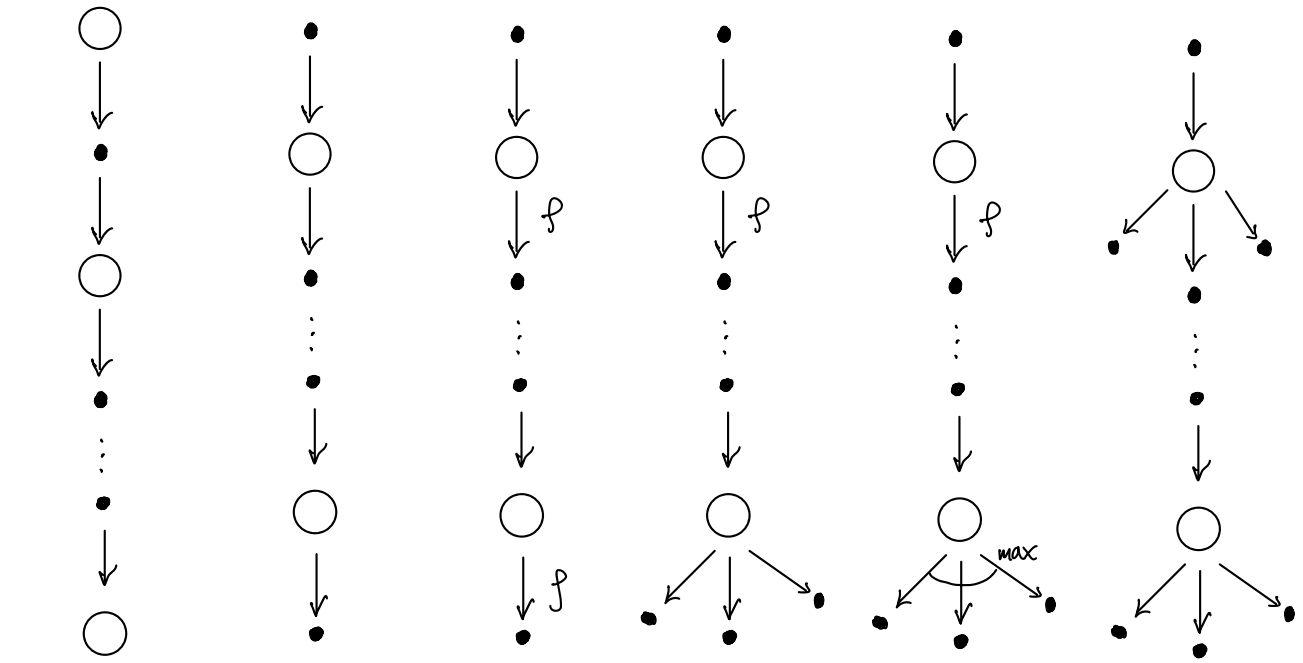
7.7: Summary

- Range of TD methods btwn one-step TD and MC methods.
- n-step methods look ahead to the next n-rewards, states, and actions.
- All n-step methods require a delay of n-steps before updating (because we need to know what happens in the next n-steps) ⟶ Eligibility traces.
- They also involve more computation than previous methods (there is always more computation beyond the one-step methods, but it's generally worth it since one-step is limited).
- Two approaches for off-policy learning have been explained:
  - one with importance sampling is simple but has high variance,
  - The other is based on tree-backup updates and is the natural extension of Q-learning to multistep case with stochastic target policies.

TD(0): high bias, low variance,

n step TD: medium bias, medium variance,

MC: 0 bias, high variance.



n-step TD on-policy prediction. Estimating V.

n-step SARSA on-policy Control. estimating q

n-step SARSA off-policy Control. estimating q

n-step Expected SARSA off-policy Control. estimating q

n-step Q-learning. off-policy control.

n-step tree-backup. off-policy control.

Targets:

n-step SARSA: $R_t + \gamma R_{t+1} + \cdots + \gamma^{n-1} R_{t+n-1} + \gamma^n Q(S_{t+n}, a_{t+n})$

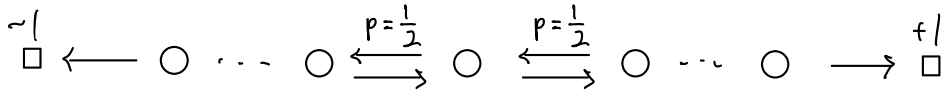
n-step off-policy SARSA: $\underbrace{\rho_{t:t+n-1}}_{\text{only difference}} \left[R_t + \gamma R_{t+1} + \cdots + \gamma^{n-1} R_{t+n-1} + \gamma^n Q(S_{t+n}, a_{t+n}) \right]$

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

$Q(\sigma)$: $R_{t+1} + \gamma \pi(A_{t+1}|S_{t+1}) \left[G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1}) \right] + \gamma \bar{V}_{h-1}(S_{t+1})$

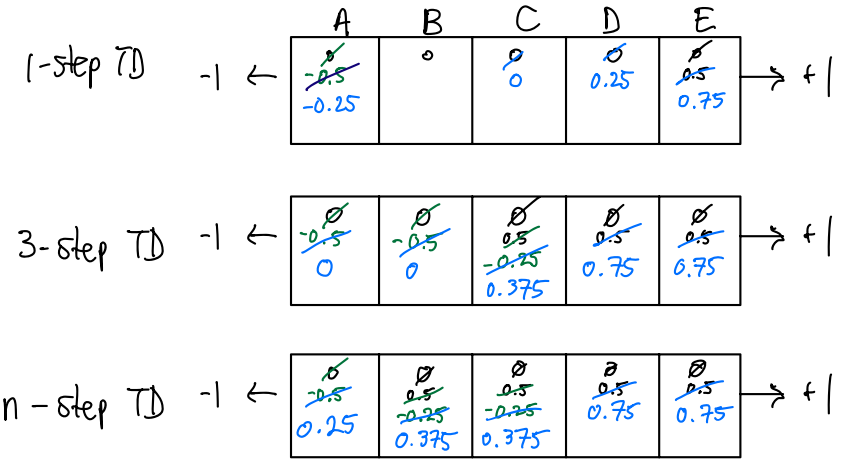
Example in the lecture:

Chain Markov Reward Process (length = 19)



Mini 5-chain $\alpha = 0.5$ $\gamma = 1$

True V(s) $-\frac{2}{3}$ $-\frac{1}{3}$ 0 $\frac{1}{3}$ $\frac{2}{3}$



Experiences

✓ B, C, D, E, +1

✓ C, B, A, -1

✓ A, B, C, D, E, +1

$$V_{t+n}(S_t) \leftarrow V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]$$

MC can be high variance.