Reinforcement Learning

- Sarath Chandar

6. Temporal Difference Learning

6. Temporal-Difference Learning

TD Learning - a combination of MC and DP.

→ Like MC, TD methods can directly learn from experience without accessing to the dynamics of the world.

→ Like DP, TD methods can bootstrap their estimates without waiting for the episode to end.

TD Prediction:-

A simple every-visit Monte Carlo method suitable for non-stationary env:

Constant-α MC

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

where G_t is the actual return,

α is a constant step-size parameter

MC methods wait until the end of episode to determine the increment to $V(S_t)$.

TD methods wait only one time step.

$$V(S_t) = V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

Target for MC : G_t

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

Tabular TD for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \quad \text{---(1)}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right]$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right] \quad \text{---(2)}$$

MC methods use (1) as target while TD methods use (2) as the target.

Note: 1. MC target is an estimate because the expected value of ① is not known.

2. DP target is an estimate because $v_\pi(S_{t+1})$ is not known and the current estimate $V(S_{t+1})$ is used instead.

2. TD target is an estimate for both reasons: It samples the expected value in ② and uses the current estimate of V instead of v_π.

Backup diagram for TD:

TD updates are sample updates.

TD error: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

TD error depends on the next state and reward. Hence it is not available until one time step later.

Note that is V does not change during the episode (as in MC methods), then MC error can be written as sum of TD errors:

$$G_t - V(S_t) = R_{t+1} + \gamma G_{t+1} - V(S_t)$$

$$= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1})$$

$$= \underline{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)} + \gamma G_{t+1} - \gamma V(S_{t+1})$$

$$= \delta_t + \gamma (G_{t+1} - V(S_{t+1}))$$

$$= \delta_t + \gamma \delta_{t+1} + \gamma^2 (G_{t+2} - V(S_{t+2}))$$

$$= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \dots + \gamma^{T-t-1} \delta_{T-1}$$

$$+ \gamma^{T-t} (G_T - V(S_T))$$

(G_T: 0, $V(S_T)$: 0)

$$= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k .$$

Note: This identity is not exact if V is updated during the episode (as in TD), but if step size is small, it may hold approximately.

Example: Driving home:

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

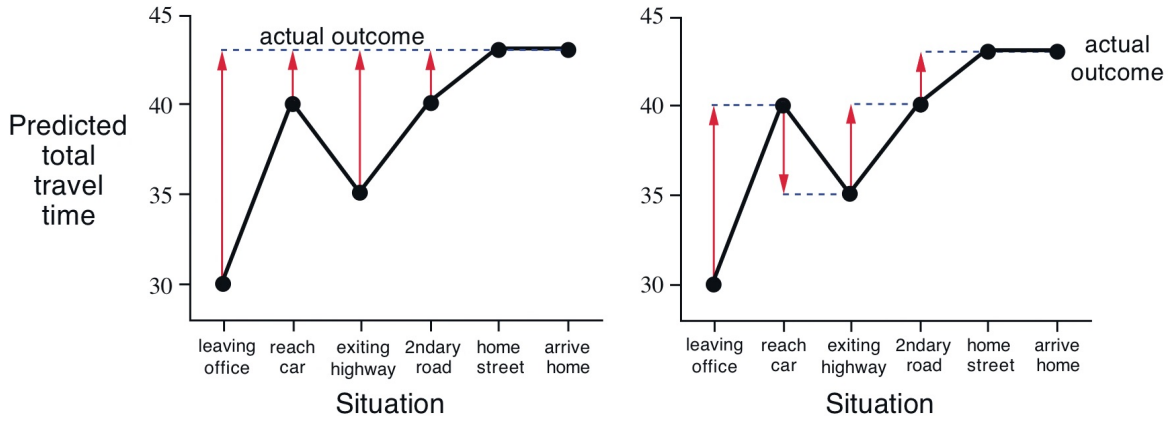
Rewards - elapsed time on each leg of the journey.

$\gamma = 1$. No discounting.

return from each State = actual time to go from that State.

Value of each State = expected time to go.

Second coln: Estimated state value.



MC

TD.

Advantages of TD prediction methods :-

TD methods learn a guess from a guess.

(1) When compared to DP methods, they do not require the model of the environment.

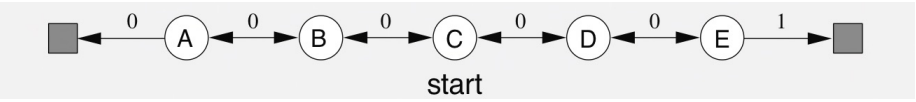
(2) When compared to MC methods, they are implemented in an online, fully incremental fashion. Useful in applications with very long episodes and continuing tasks.

MC has to ignore episodes in which exploratory actions are taken, which can greatly slow learning. TD methods learn from each transition.

For any fixed policy π, TD 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.

Random Walk Example :-

Consider the following Markov Reward Process (MRP) (an MDP without actions).

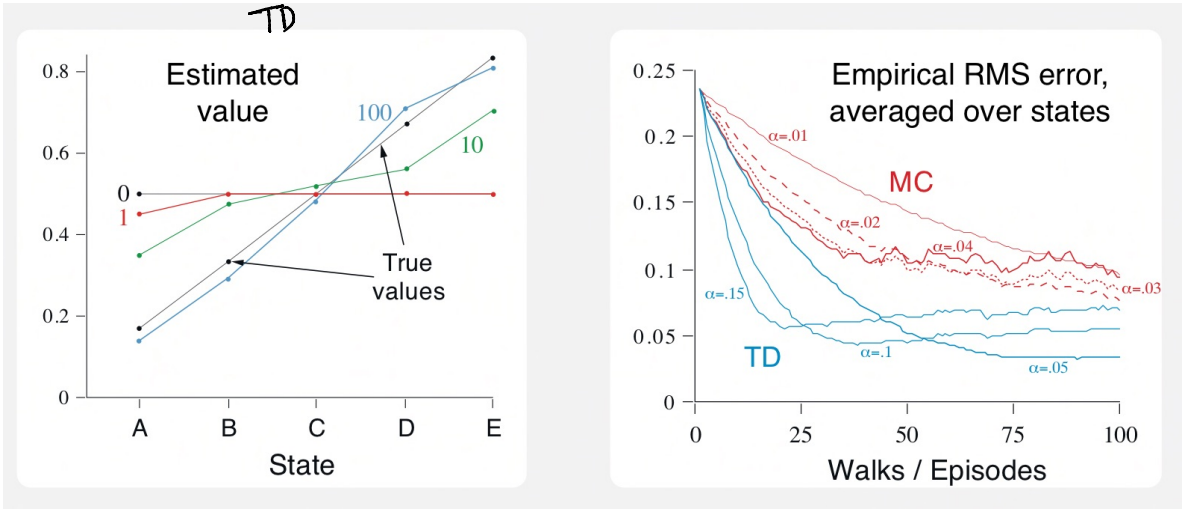


undiscounted task.

True value of states A to E : probability of terminating on the right starting from that state.

A to E : $1/6, 2/6, 3/6, 4/5, 5/6$.

initial state value of 0.5 for all s.



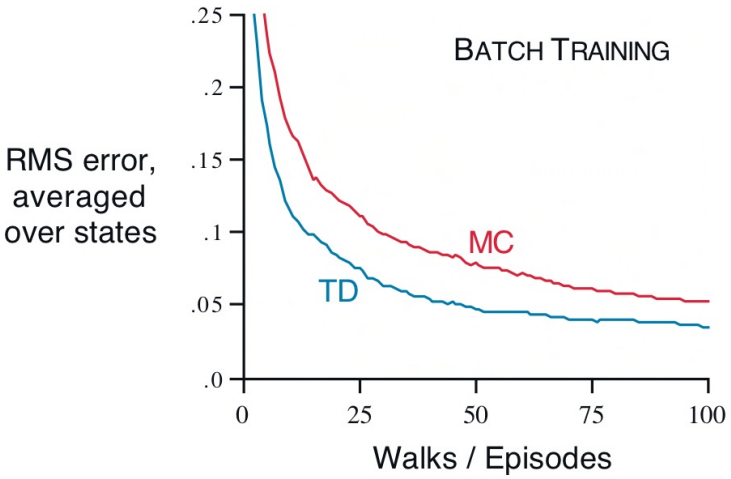
Optimality of TD :

Let us consider batch-updating

Setting where all the experience (episode) collected so far is presented repeatedly to the algorithm.

Under batch updating, TD converges deterministically to a single answer independent of α (provided it is sufficiently small). Constant α-MC method also converges deterministically, but to a different solution!

Random walk with batch updating :-



Constant α MC converges to values that are sample averages of the actual returns experienced. These are optimal in the sense that they minimize RMSE

from the actual returns in the training set.

Then how can TD perform better than MC?

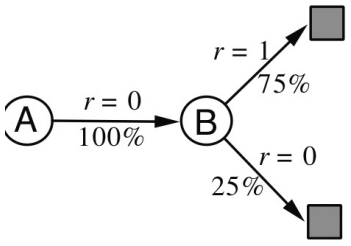
Consider the following episodes from an unknown MRP.

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

What is V(A), V(B)?

$$V(B) = \frac{3}{4}.$$

What is V(A)? There are two reasonable answers:

① 

$V(A) = 3/4$.

Here we first model the MRP.

② $V(A) = 0$. This is what MC will predict.

Note: this answer will get zero training error. But if the process is Markov, ans ① will have less future error!

Batch MC - find the estimates that minimize mean-squared error on the training set.

Batch TD - Find the estimates that would be exactly correct for the maximum likelihood model of the Markov process.

TD first models the Markov process based on the observed transitions and given this model, attempt to compute the estimate of the value fn that would be exactly correct if the model were exactly correct.

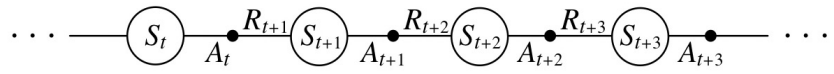
↳ certainty-equivalence estimate

It is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximated.

Batch TD converges to the certainty-equivalence estimate.

This explains why TD may converge quicker than MC.

Sarsa : On-policy TD Control :-



We want to estimate $q_\pi(s,a)$

We can extend TD for $v_\pi(s)$ to TD for $q_\pi(s,a)$.

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

If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1}) = 0$.

This rule uses every element of $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ and hence the name Sarsa.

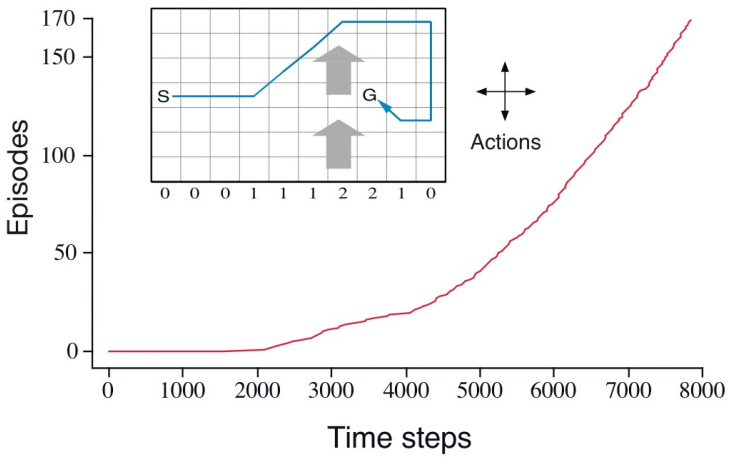
Sarsa backup diagram.

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 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Windy gridworld example:-



wind - resultant next states are shifted upward based on strength.

- undiscounted, episodic.
- reward of −1 until goal is reached.

ϵ-greedy Sarsa with $\epsilon = 0.1$, $\alpha = 0.5$, inital $Q(S,A) = 0$.

Note: MC cannot be used here since termination is never guaranteed. On the other hand, TD can learn within the episode.

Q-learning : Off-Policy TD Control (Watkins, 1989)

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

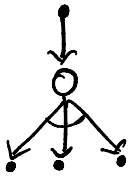
Here Q directly approximates q_*, independent of the policy being followed.

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

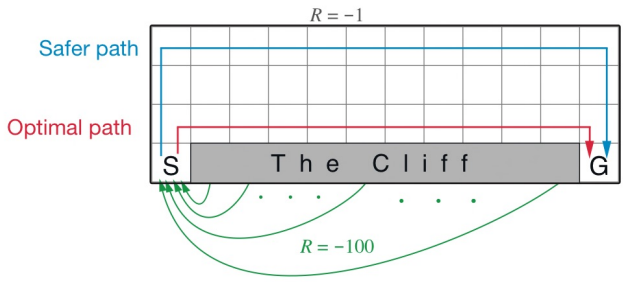
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-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

Backup diagram for Q-learning :-

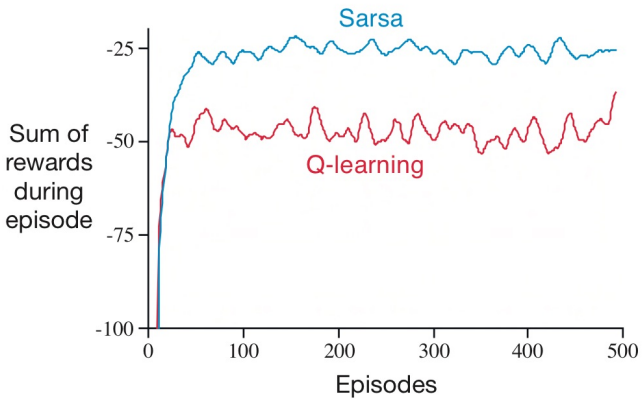


Example: cliff walking :-



- undiscounted, episodic.

- Q-learning learns the optimal policy. But this results in occassionally falling off the cliff - due to ϵ-greedy.



- Sarsa learns a longer but safer path.

Expected Sarsa:-

Consider an algo. that has the following update rule

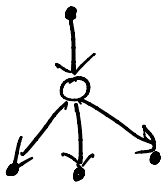
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma E_\pi [Q(S_{t+1}, A_{t+1}) | S_{t+1}] - Q(S_t, A_t) \Big]$$

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

but that otherwise follows the schema of Q-learning.

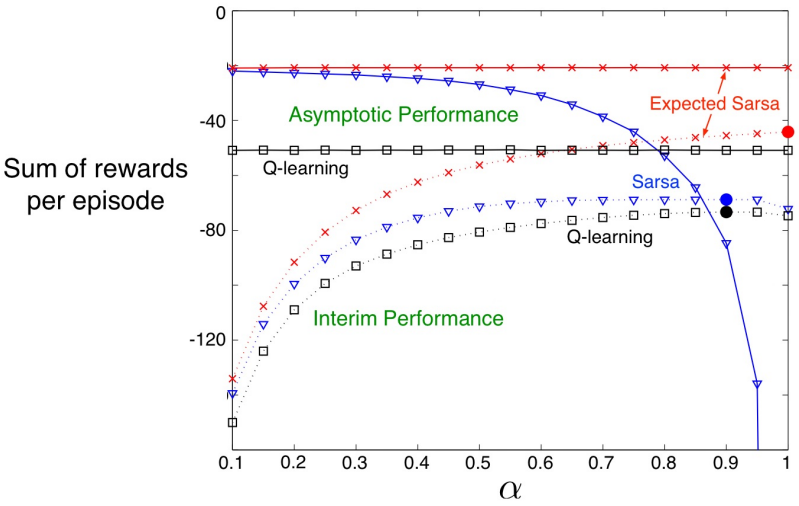
Given the next state S_{t+1}, this algorithm moves deterministically in the same direction as Sarsa moves in expectation and hence called Expected Sarsa.

Backup diagram:



- More Computation than Sarsa, but eliminates the variance due to the random selection of A_{t+1}.
- Performs better than Sarsa and Q-learning.

Cliff walking example



Here we used on-policy expected Sarsa. But one can define off-policy expected Sarsa.

If π is greedy and behavior policy is exploratory, then expected Sarsa is exactly Q-learning!

Maximization bias and double learning:-

All control algorithms so far involve maximization in the construction of their target policies.

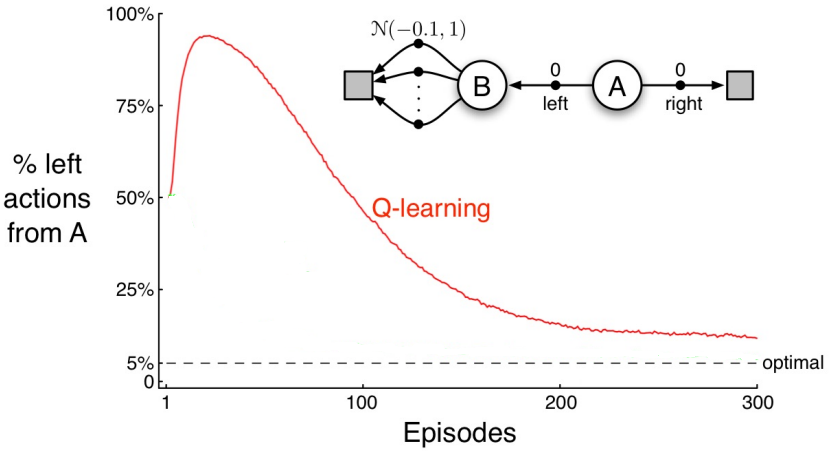
Ex: Q-learning - target policy is greedy policy defined with max.

Sarsa - Policy is ϵ-greedy which has max.

Note: A maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to significant positive bias.

Consider a single state s, where all actions have $q(s,a)=0$. But the estimates $Q(s,a)$ are uncertain and thus distributed above and below 0. The max of the estimate is positive, a positive bias / Maximization bias.

Example:-



How to avoid maximization bias?

Consider the bandit case. One way to view the problem is that it is due to using the same samples both to determine the maximizing action and to estimate its value.

Suppose we divided the trials in two sets and used them to estimate $Q_1(a)$ and $Q_2(a)$, each an estimate of the true value $q(a)$.

Use Q_1 for max: $A^* = \underset{a}{\operatorname{argmax}}\, Q_1(a)$

Use Q_2 for estimate: $Q_2(A^*) = Q_2\left(\underset{a}{\operatorname{argmax}}\, Q_1(a)\right)$

This estimate will be unbiased in the sense that $\mathbb{E}\left[Q_2(A^*)\right] = q(A^*)$.

We can also repeat the process with roles reversed to yield a second unbiased

estimate.

$$Q_1(\underset{a}{\text{argmax}}\ Q_2(a)).$$

This is the idea of double learning.

In each trial, only update one estimate.

Double learning — doubles the memory requirement.

— but same computational requirement.

This idea easily extends to MDP.

Double-Q-learning:-

If coin flip head,

$$Q_1(S_t, A_t) = Q_1(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q_2\left(S_{t+1}, \underset{a}{\text{argmax}}\, Q_1(S_{t+1}, a)\right) - Q_1(S_t, A_t)\right]$$

If coin flipped tail, update Q_2.

Note: behavior policy can use both action-value estimates.

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 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\textit{terminal}, \cdot) = 0$

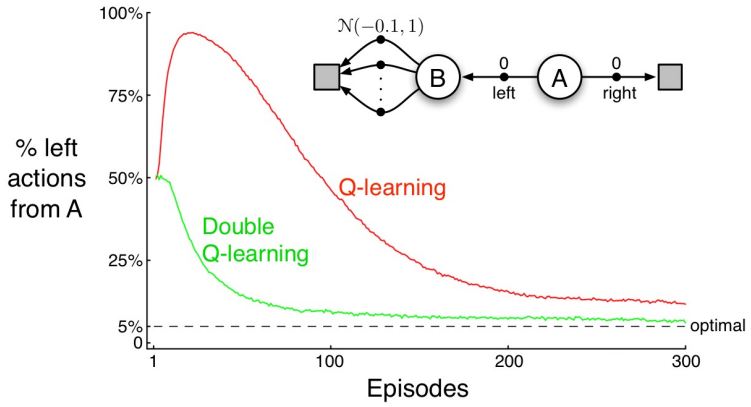
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal



---

n-step bootstrapping :-

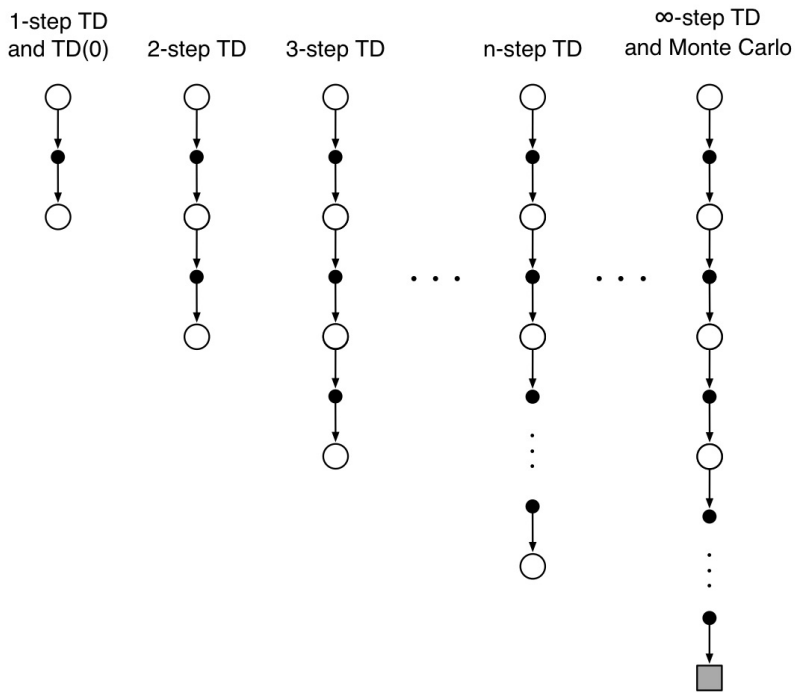
n-step TD methods → that generalize both MC and one-step TD.

With one-step TD, the same time step is used for changing action and bootstrapping.

n-step TD decouples this and have different timescales for action and bootstrapping.

n-step TD prediction :-

What are the space of methods lying between MC and TD methods?



Methods that use n-step updates are still TD methods because they still change an earlier estimate based on how it differs from a later estimate.

Consider the update of the estimated value of state S_t as a result of the state-reward sequence $S_t, R_{t+1}, S_{t+1}, R_{t+2}, \ldots R_T, S_T$

MC: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$

In MC, G_t is the target of the update.

1-step TD: $G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$

↑ truncated return for time t with $\gamma V_t(S_{t+1})$ replacing $\gamma R_{t+2} + \gamma^2 R_{t+3} \ldots + \gamma^{T-t-1} R_T$.

2-step TD: $G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$.

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

$\forall\ n, t : n \geq 1$ and $0 \leq t < T-n$.

Note: If $t+n \geq T$, then all missing terms are taken as zero.

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

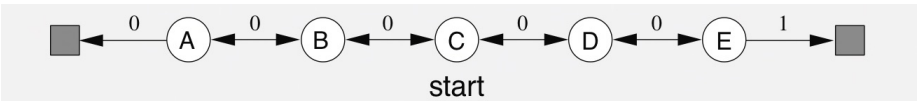
$0 \le t < T$.

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

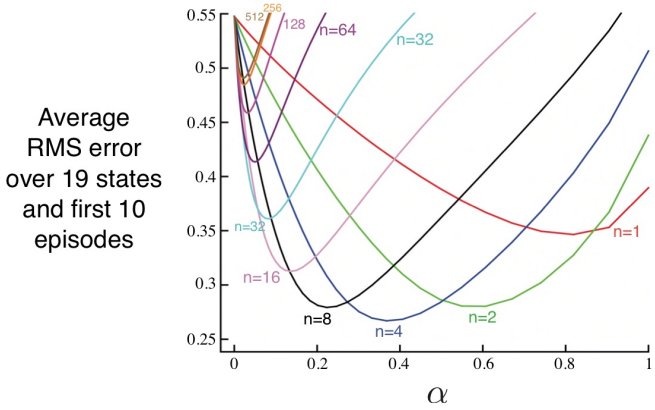
Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ 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} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

Example: n-step TD methods on the Random walk.



19 states instead of 5, with -1 outcome on left, all values initialized to 0.



n-step Sarsa :-

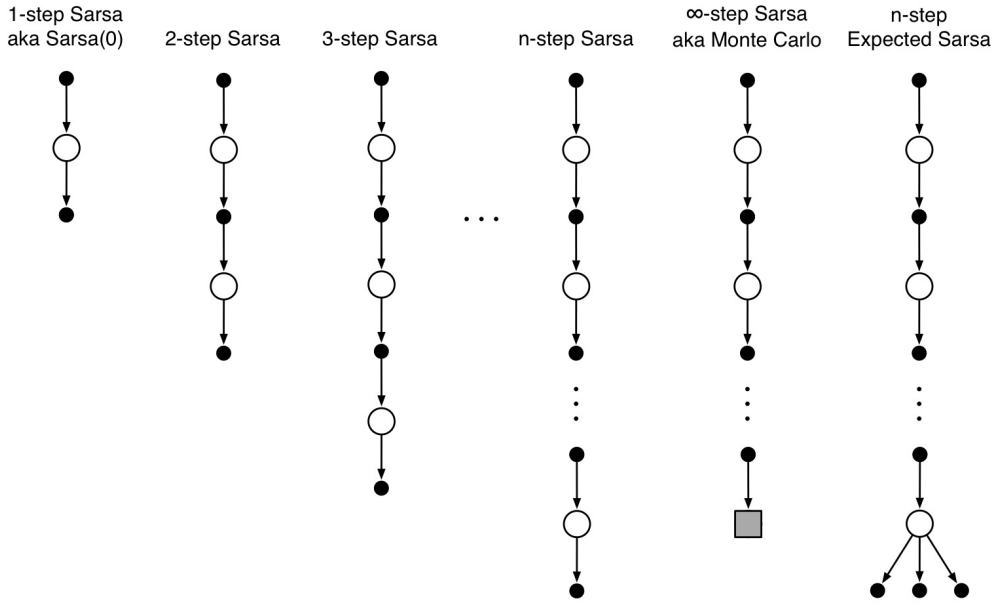
$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n})$$

$n \geq 1,\ 0 \leq t < T-n.$

with $G_{t:t+n} = G_t$ if $t+n \geq T$.

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

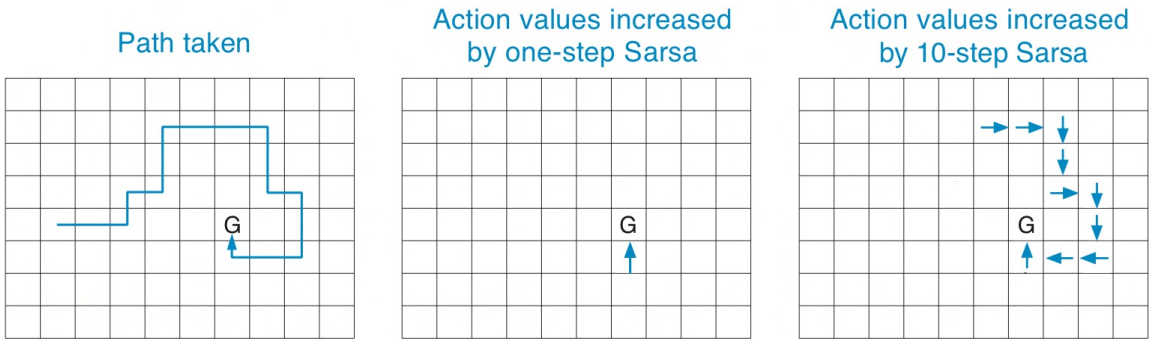
$0 \leq t < T$



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 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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ 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 π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Illustration: 1-step Sarsa vs. 10-step Sarsa.



n-step expected Sarsa:

$$G_{t:t+n} = R_{t+1} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n})$$

$t+n < T$

where $\bar{V}_t(s) = \sum_a \pi(a|s) Q_t(s,a) \quad \forall s \in \mathcal{S}$.

n-step off-policy learning :-

To make a simple off policy version of n-step TD, the update for time t can simply be weighted by $\rho_{t:t+n-1}$:

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

$0 \leq t < T$

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

If two policies are actually same, then $\rho = 1$.

off-Policy n-step Sarsa :

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

$0 \leq t < T.$

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 $s \in \mathcal{S}, a \in \mathcal{A}$
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: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$