Lecture 10 - n-step Bootstrapping

Instructor: GuangBing Yang, PhD

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

Introduction to n-step Bootstrapping

- What is n-step Bootstrapping learning?
  - It combines the Monte Carlo (MC) methods and the one-step temporal-difference (TD) methods to become a n-step TD methods
  - n-step methods are intermediate between MC and TD(0)
  - n-step methods enable bootstrapping to occur over multiple steps

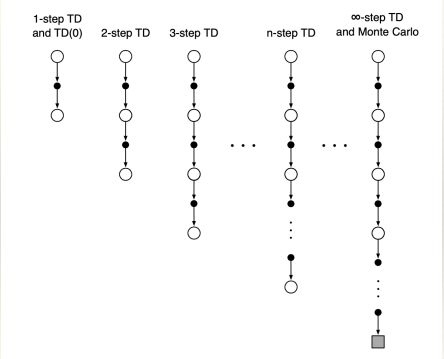
yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

n-step TD Prediction Problem

- Again, n-step TD prediction problem is also the problem of policy evaluation.
  - Monte Carlo methods need knowing the entire episode to determine the update for each state.
  - TD(0) only needs to know one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards.
  - Like the one-step TD methods, it performs an update based on an intermediate number of rewards: more than one, but not all of them until termination.
  - For example, a two-step update uses the first two rewards and the estimated value of the state two steps later. Similarly, three-step uses three rewards, and so on.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

Backup diagram of n-step TD



n-step backup diagram

n-step TD Prediction Problem

- Note that the methods that use n-step updates are still TD methods.
- They still change an earlier estimate based on how it differs from a later estimate.
- Now the later estimate is not one step later, but n steps later.
- Thus, methods in which the temporal difference extends over n steps are called n-step TD methods.
- Consider the update of the estimated value of state S_t as a result of the state-reward sequence, $S_t, R_{t+1}, S_{t+1}, R_{t+2}, \ldots, R_T, S_T$.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

n-step TD Prediction Problem

- In Monte Carlo methods, the returns are given as:
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$
- In TD(0), which is the one-step methods, the returns are given as:
  - $G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$
- In TD(2), which is the two-step methods, the returns are given as:
  - $G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$
- In TD(n), which is the n-step methods, the returns are given as:
  - $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \forall n, t, \; n \geq 1, \; 0 \leq t < T - n$

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

n-step TD Prediction Problem

- Note that n-step returns for n > 1 involve future rewards and states are not available at the time of transition from t to t+1.
- $V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)], \; 0 \leq t < T,$
  - while the values of all other states remain unchanged: $V_{t+n}(s) = V_{t+n-1}(s), \; \forall s \neq S_t.$
- This algorithm is called n-step TD.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

n-step TD prediction algorithm

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$
  Until $\tau = T - 1$

Error reduction property of n-step

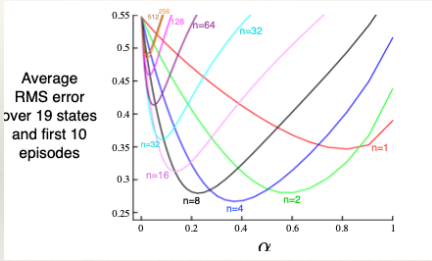
- The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}.
- Note the worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V_{t+n-1}:
  $\max_s |E_\pi[G_{t:t+n} | S_t = s] - v_\pi(s)| \leq \gamma^n \max_s |V_{t+n-1}(s) - v_\pi(s)|$
- This is called error reduction property of n-step returns.
- Because this property, all n-step TD methods converge to the correct predictions under appropriate technical conditions.
- The n-step TD methods thus form a family of sound methods, with one-step TD methods and Monte Carlo methods as extreme members.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021
9

9

Example of n-step TD methods - Random Walk

- In TD error -- the difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$.
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- Note that the TD error at each time is the error in the estimate made at that time. Because the TD error depends on the next state and next reward, it is not actually available until one time step later.
- That is δ_t is the error in $V(S_t)$, available at time t+1.
- $G_t - V(S_t) = R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k$
- Which is the sum of TD errors if the array V does not change during the episode.



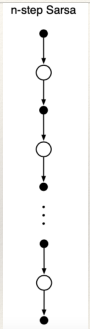
Performance of n-step TD methods as a function of α for various values of n, on a 19-state random walk task

10

10

n-step Sarsa–TD control

- The main idea is to simply switch states for actions (state-action pairs) and then use an ϵ-greedy policy
- The return function of n-step salsa TD is given as:
  - $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$,
  - $\forall n \geq 1,\ t, n \geq 1,\ 0 \neq t < T - n$, with $G_{t:t+n} = G_t$ if $t + n >= T$.
- The backup diagram of n-step salsa TD is shown to the right:



n-step Sarsa backup diagram

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021
11

11

n-step Sarsa–TD control

- The natural algorithm is then given as:
  - $Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$
- While the values of all other states remain unchanged:
  - $Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$
    - for all s, a such that $s \neq S_t$ or $a \neq A_t$.
- This is the algorithm called n-step Sarsa.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021
12

12

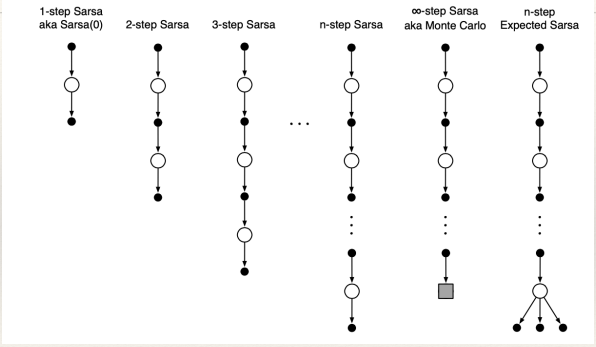
n-step Sarsa–TD control

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

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

Backup n-step Sarsa–TD control



The backup diagrams for the spectrum of n-step methods for state–action values, range from the one-step update of Sarsa(0) to the up-until-termination update of the Monte Carlo method.

n-step Expected Sarsa–TD control

- Another approach is expected Sarsa, rather than based on n steps of real rewards, the expected Sarsa estimated value of the nth next state–action pair.
  - Expected Sarsa: $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n})$, with $(G_{t:t+n} = G_t \text{ for } t+n \geq T)$, while $\bar{V}_t(s)$ is the expected approximate value of state s, using the estimated action values at time t, under the target policy:
  - $\bar{V}_t(s) = \sum_s \pi(a \mid s) Q_t(s,a), \forall s \in S$
  - If s is terminal, then its expected approximate value is defined to be 0.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

n-step Off-policy learning

- The off-policy learning is learning the value function for one policy π following another policy b, π is the greedy policy for the current-value function estimate, and b is a more exploratory policy, ϵ-greedy.
- The relative probability of taking the actions is used to get the estimated value over n steps.
- The update for time t can simply be weighted by $\rho_{t:t+n-1}$:
- $V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)]$

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

n-step Off-policy learning

- $V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)]$,
- where $\rho_{t:t+n-1}$ is called the importance sampling ratio, is the relative probability under the two policies of taking the n actions from A_t to A_{t+n-1}.
- $\rho_{t:h} = \prod_{k=t}^{min(h,T-1)} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$.

n-step Off-policy learning

- For example,
  - if any action does not be taken by π (i.e., $\pi(A_k \mid S_k) = 0$) then the n-step return should be given zero weight and be totally ignored.
  - if an action is taken that π, it takes with much greater probability than b does, then this will increase the weight that would otherwise be given to the return.
- This makes sense because that action is characteristic of π (to be learned) but is selected only rarely by b and thus rarely appears in the data.
- To emphasize this occasional action, the algorithm needs over-weight it when it does occur. Note that if the two policies are actually the same (the on-policy case) then the importance sampling ratio is always 1.

n-step Off-policy learning

- Thus the update $V_{t+n}(S_t)$ generalizes and can completely replace the earlier n-step TD update.
- Similarly, the n-step Sarsa update discussed in previous slide can be completely replaced by a simple off-policy form:
  - $Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n-1}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \forall 0 \le t < T$
  - Note that the importance sampling ratio here starts and ends one step later than for n-step TD. This is because here we are updating a state–action pair.

n-step Off-policy learning—Algorithm

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 $\tau \geq 0$:
      $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:t+n-1}$)
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

Per-decision methods with Control Variates

- The multi-step off-policy methods are simple and conceptually clear.
- They are probably not the most efficient.
- A more sophisticated approach would use per-decision importance sampling ideas such as were introduced in Monte Carlo per-decision importance sampling.
- If recursively write the n-step return as: $G_{t:h} = R_{t+1} + \gamma G_{t+1:h}, \;\; t < h < T,$ where $G_{h:h} = V_{h-1}(S_h)$

Per-decision methods with Control Variates

- The effect of following a behavior policy b that is not the same as the target policy π.
- All of the resulting experience, 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, $\rho_{t:h} = \dfrac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$
- Suppose the action at time t would never be selected by π, so that ρ_t is zero. Then a simple weighting would result in the n-step return being zero, which could result in high variance when it was used as a target.

Per-decision methods with Control Variates

- Instead, in this more sophisticated approach, one uses an alternate, off-policy definition of the n-step return ending at horizon h, as:
- $G_{t:h} = \rho_t(R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V_{h-1}(S_t), \;\; t < h < T$
- where again $G_{h:h} = V_{h-1}(S_h)$
- In this approach, if ρ_t is zero, then instead of the target being zero and causing the estimate to shrink, the target is the same as the estimate and causes no change.
- The importance sampling ratio being zero means we should ignore the sample, so leaving the estimate unchanged seems appropriate.

Per-decision methods with Control Variates

- The term $(1 - \rho_t)V_{h-1}(S_t)$ is control variate.
- It does not change the expected update;
- The importance sampling ratio has expected value one ($\sum_a \pi(a \mid S_k) = 1$)and is uncorrelated with the estimate, so the expected value of the control variate is zero.
- Also note that the off-policy definition is a strict generalization of the earlier on-policy definition of the n-step return, as the two are identical in the on-policy case, in which ρ_t is always 1.

Per-decision methods with Control Variates

- 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.
- That first action is the one being learned; it does not matter if it was unlikely or even impossible under the target policy—it has been taken and now full unit weight must be given to the reward and state that follows it.
- Importance sampling will apply only to the actions that follow it.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

25

Per-decision methods with Control Variates

- An off-policy form with control variates is given as:
- $G_{t:h} = R_{t+1} + \gamma(\rho_{t+1}G_{t+1:h} + \bar{V}_{h-1}(S_{t+1}) - \rho_{t+1}Q_{h-1}(S_{t+1}, A_{t+1}))$, or
- $G_{t:h} = R_{t+1} + \gamma\rho_{t+1}(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) + \gamma\bar{V}_{h-1}(S_{t+1}), \;\; t < h < T$
  - if h < T, the recursion ends with $G_{h:h} = Q_{h-1}(S_h, A_h)$
  - if h >= T, the recursion ends with $G_{h:h} = R_T$

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

26

The n-step Tree Backup Algorithm

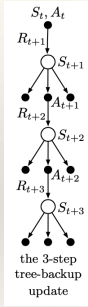
- It is a kind of off-policy learning without importance sampling.
- Off-policy learning is possible without importance sampling,
- Q-learning and Expected Sarsa do this for the one-step case
- For n-step, tree-backup algorithm does the same thing.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

27

The n-step Tree Backup Algorithm

- Backup diagrams of the n-step tree backup diagram is shown to the right
- It is an update from the entire tree of estimated action values.
- The update is from the estimated action values of the leaf nodes of the tree.
- Each leaf node contributes to the target with a weight proportional to its probability of occurring under the target policy π



the 3-step tree-backup update

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

28

The n-step Tree Backup Algorithm

- Each first-level action a contributes with a weight of $\pi(a \mid S_{t+1})$, except that the action actually taken, A_{t+1}, does not contribute at all.
- Its probability, $\pi(A_{t+1} \mid S_{t+1})$, is used to weight all the second-level action values.
- Each third-level action contributes with weight $\pi(A_{t+1} \mid S_{t+1})\pi(A_{t+2} \mid S_{t+2})\pi(a'' \mid S_{t+3})$, and so on.
- It is as if each arrow to an action node in the diagram is weighted by the action's probability of being selected under the target policy and, if there is a tree below the action, then that weight applies to all the leaf nodes in the tree.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

29

29

The n-step Tree Backup Algorithm

- The general recursive definition of the tree-backup n-step return:
- $G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+n}$
- For t < T -1, n >= 2, with n=1 case handled by $G_{t:t+1} = R_{t+1} + \gamma \sum_{a} \pi(a \mid S_{t+1}) Q_t(S_{t+1}, a)$, except for $G_{T-1:t+n} = R_T$

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

30

30

The n-step Tree Backup Algorithm

- The detailed equations for the n-step tree-backup algorithm.
  - The one-step return (target) is the same as that of Expected Sarsa, $G_{t:t+1} = R_{t+1} + \gamma \sum_{a} \pi(a \mid S_{t+1}) Q_t(S_{t+1}, a)$, for t < T - 1,
- and the two-step tree-backup return is: $G_{t:t+2} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1})(R_{t+2} + \gamma \sum_{a} \pi(a \mid S_{t+2}) Q_{t+1}(S_{t+2}, a))$
- $G_{t:t+2} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:t+2}$, for t < T-2

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

31

31

The n-step Tree Backup Algorithm

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

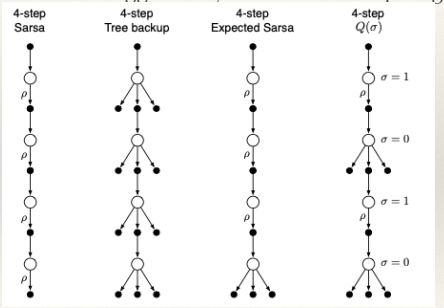
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations can take their index mod $n+1$

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

32

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

- This is the for unification is suggested by the fourth backup diagram shown to the below.



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

- This is the idea that one might decide on a step-by-step basis whether one wanted to take the action as a sample, as in Sarsa, or consider the expectation over all actions instead, as in the tree-backup update. Then, if one chose always to sample, one would obtain Sarsa, whereas if one chose never to sample, one would get the tree-backup algorithm. Expected Sarsa would be the case where one chose to sample for all steps except for the last one.
- To increase the possibilities even further we can consider a continuous variation between sampling and expectation. Let $\sigma_t \in [0,1]$ denote the degree of sampling on step t, with $\sigma = 1$ denoting full sampling and $\sigma = 0$ denoting a pure expectation with no sampling.
- The random variable σ_t might be set as a function of the state, action, or state-action pair at time t. This algorithm is called n-step $Q(\sigma)$.

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

- $G_{t:h} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1}) Q_{h-1}(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:h}$
- $G_{t:h} = R_{t+1} + \gamma \bar{V}_{h-1}(St+1) - \gamma \pi(A_{t+1} \mid S_{t+1}) Q_{h-1}(S_{t+1}, A_{t+1}) + \gamma \pi(A_{t+1} \mid S_{t+1}) G_{t+1:h}$
- $G_{t:h} = R_{t+1} + \gamma \pi(A_{t+1} \mid S_{t+1})(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) + \gamma \bar{V}_{h-1}(S_{t+1})$
- It is exactly like the n-step return for Sarsa with control variates except with the action probability $\pi(A_{t+1} \mid S_{t+1})$ replace the importance-sampling ratio ρ_{t+1}.
- For $Q(\sigma)$, there is:
  $G_{t:h} = R_{t+1} + \gamma(\sigma_{t+1}\rho_{t+1} + (1 - \rho_{t+1})\pi(A_{t+1} \mid S_{t+1}))(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})) + \gamma \bar{V}_{h-1}(S_{t+1})$

Recap

- In this lecture we discussed several temporal-difference learning methods that lie in between the one-step TD methods of the previous chapter and the Monte Carlo methods of the chapter before.
- Methods that involve an intermediate amount of bootstrapping are important because they will typically perform better than either extreme.
- n-step methods, which look ahead to the next n rewards, states, and actions.
- The state-value update shown is for n-step TD with importance sampling, and the action-value update is for n-step Q(\sigma), which generalizes Expected Sarsa and Q-learning.

Recap

- A further drawback is that they involve more computation per time step than previous methods.
- Compared to one-step methods, n-step methods also require more memory to record the states, actions, rewards, and sometimes other variables over the last n time steps.
- Two approaches to off-policy learning in the n-step case. One, based on
- Importance sampling is conceptually simple but can be of high variance.
- If the target and behavior policies are very different it probably needs some new algorithmic ideas before it can be efficient and practical.
- The tree-backup updates, is the natural extension of Q-learning to the multi-step case with stochastic target policies.
- It involves no importance sampling but, again if the target and behavior policies are substantially different, the bootstrapping may span only a few steps even if n is large.

yguangbing@gmail.com, Guang.B@chula.ac.th Mar 30 2021

Questions and Lab