Reinforcement Learning

n-step Bootstrapping

U Kang
Seoul National University

In This Lecture

- Unification of MC and TD methods
- n-step methods

Overview

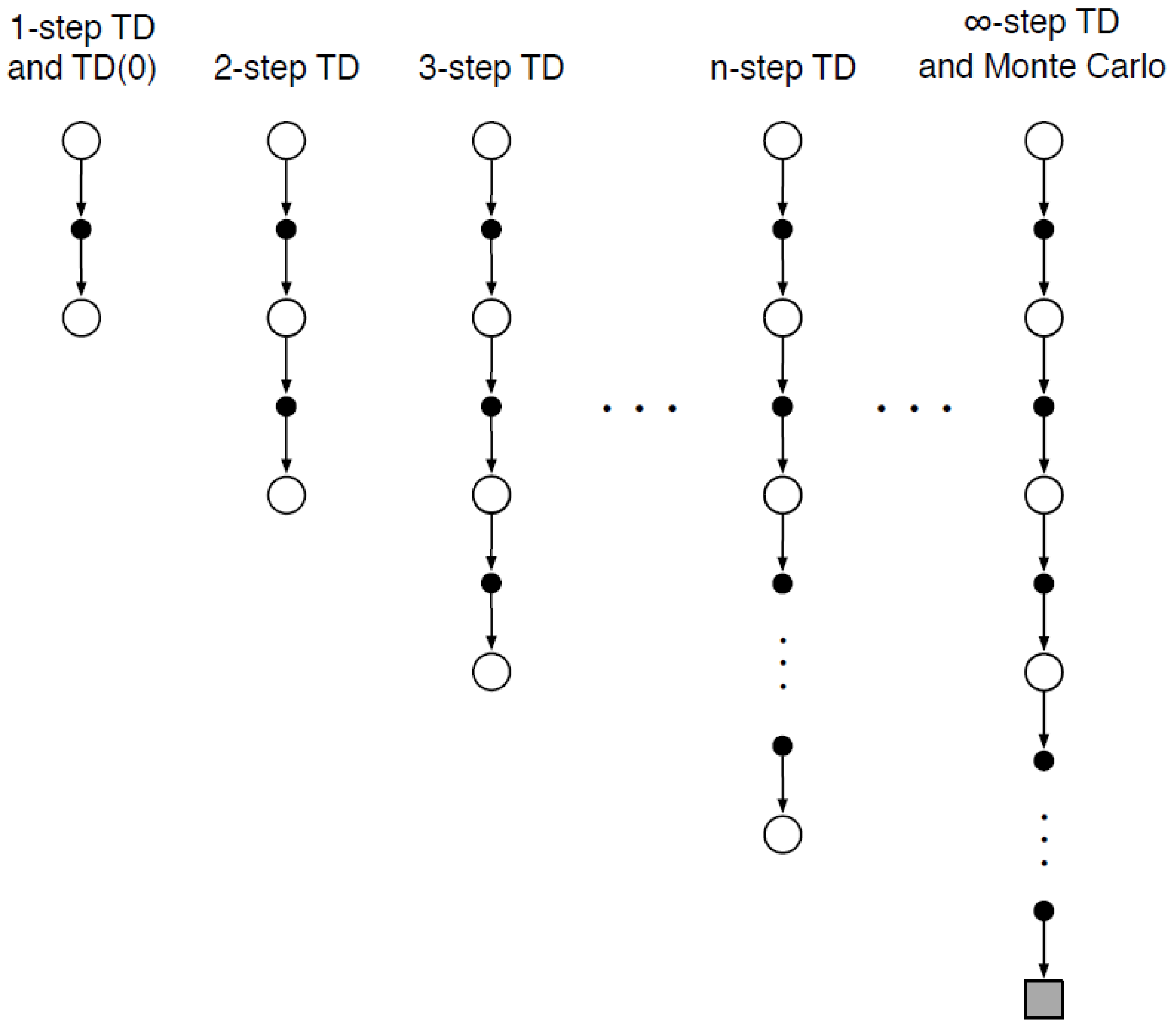
- Goal: unify the Monte Carlo (MC) methods and the one-step temporal difference (TD) methods
- Neither MC methods nor one-step TD methods are always the best
- n-step TD methods generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task
- n-step methods span a spectrum with MC methods at one end and one-step TD methods at the other; the best methods are often intermediate between the two extremes

Outline

n-step TD Prediction

- What methods lie between MC and TD methods?
- Consider estimating v_π from sample episodes generated using π
- MC methods perform an update for each state based on the entire sequence of observed rewards from that state until the end of the episode
- The update of one-step TD methods, on the other hand, is based on just the one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards
- An intermediate method would perform an update based on an intermediate number of rewards: more than one, but less than all of them until termination
- E.g., a two-step update would be based on the first two rewards and the estimated value of the state two steps later

n-step TD Prediction



Sutton and Barto, Reinforcement Learning, 2018

n-step TD Prediction

- n-step TD methods: temporal difference extends over n steps
- n-step TD methods 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 updates the estimate of $v_\pi(S_t)$ in the direction of the complete return:

$$\underline{G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} \mathrm{R_T}}$$

Target of the update

- In MC update, the target is the return

n-step TD Prediction

- In one-step TD, the target is the one-step return: the first reward plus the discounted estimated value of the next state

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

  - V_t is the estimate at time t of v_π
  - $G_{t:t+1}$ is a truncated return for time t using rewards up until time t+1, with the discounted estimate $\gamma V_t(S_{t+1})$ taking the place of the other terms $\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ of the full return
- The target for a two-step update is the two-step return:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- The target for an arbitrary n-step update is the n-step return:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

  where $n \geq 1$ and $0 \leq t < T - n$
  - If $t + n \geq T$, then all the missing terms are taken as zero, and the n-step return is equal to the ordinary full return ($G_{t:t+n} = G_t$ if $t + n \geq T$)

n-step TD Prediction

- Note that n-step returns for n > 1 involve future rewards and states that are not available at the time of transition from t to t + 1. No real algorithm can use the n-step return until after it has seen R_{t+n} and computed V_{t+n-1}
- The first time these are available is t + n; the natural state-value learning algorithm for using n-step returns is

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

while the values of all other states remain unchanged: $V_{t+n}(s) = V_{t+n-1}(s)$ for all $s \neq S_t$

- Note that no changes at all are made during the first n − 1 steps of each episode; to make up for that, an equal number of additional updates are made at the end of the episode, after termination and before starting the next episode

n-step TD Prediction

Sutton and Barto, Reinforcement Learning, 2018

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 $s \in \mathcal{S}$
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 $T \leftarrow t+1$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ 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 $\tau = T - 1$
```

Let n=3, T=6

t	=	0	1	2	3	4	5	6
τ	=			0	1	2	3	4 5

n-step TD Prediction

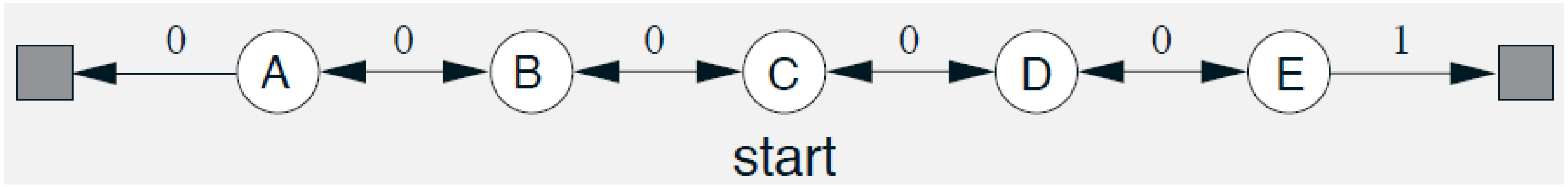
- The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}
- Error reduction property of n-step returns: their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense; the worst error of the expected n-step return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

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

- Because of the error reduction property, all n-step TD methods can be shown to converge to the correct predictions

Example: n-step TD Methods on the Random Walk

Sutton and Barto, Reinforcement Learning, 2018

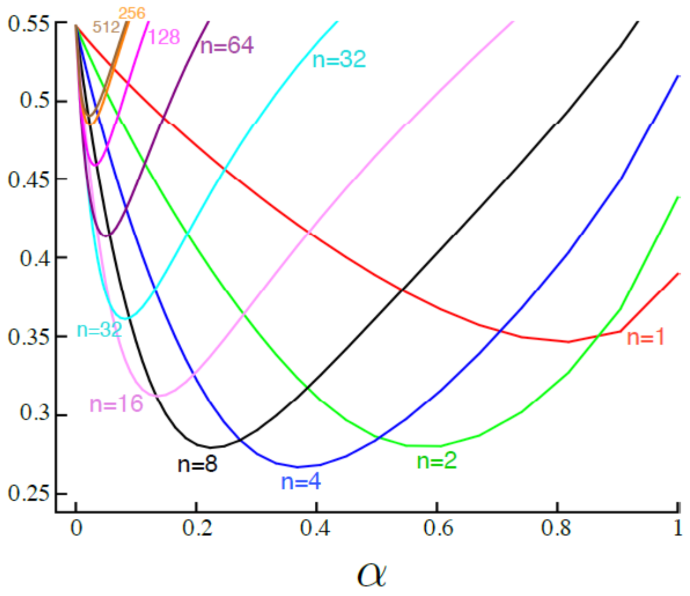


- Suppose the first episode progressed directly from the center state, C, to the right, through D and E, and then terminated on the right with a return of 1
- The estimated values of all the states started at an intermediate value, V (s) = 0.5
- As a result of this experience, a one-step method would change only the estimate for the last state, V (E), which would be incremented toward 1, the observed return
- A two-step method, on the other hand, would increment the values of the two states preceding termination: V (D) and V (E) both would be incremented toward 1
- A three-step method, or any n-step method for n > 2, would increment the values of all three of the visited states toward 1, all by the same amount

Example: n-step TD Methods on the Random Walk

- Which value of n is better?
- Results of a simple empirical test for a larger random walk process, with 19 states instead of 5
- The performance measure for each parameter setting, shown on the vertical axis, is the square-root of the average squared error between the predictions at the end of the episode for the 19 states and their true values, then averaged over the first 10 episodes and 100 repetitions of the whole experiment
- Methods with an intermediate value of n worked best

Average RMS error over 19 states and first 10 episodes



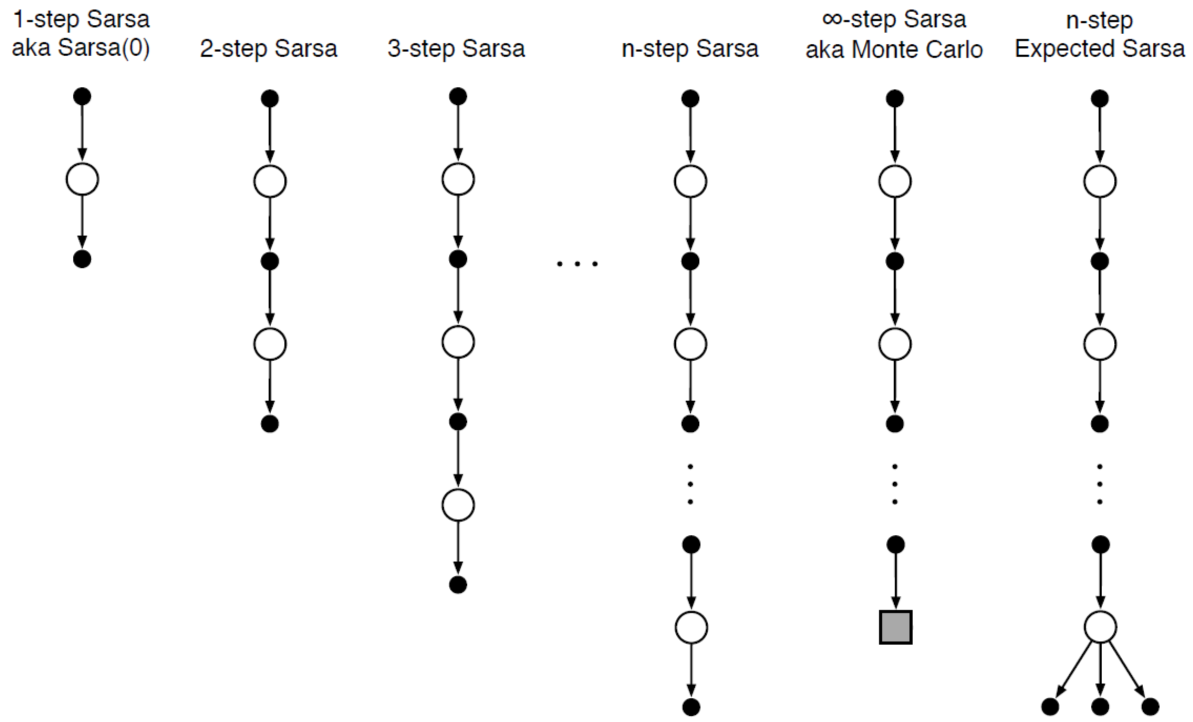
Sutton and Barto, Reinforcement Learning, 2018

Outline

- ☑ n-step TD Prediction
- ➡ ☐ **n-step Sarsa**
- ☐ n-step Off-policy Learning
- ☐ n-step Tree Backup
- ☐ Conclusion

n-step Sarsa

- How can n-step methods be used not just for prediction, but for control?
- n-step Sarsa: the n-step version of Sarsa
- Main idea of n-step Sarsa: simply switch states for actions (state–action pairs) and then use an ϵ-greedy policy



Sutton and Barto, Reinforcement Learning, 2018

n-step Sarsa

- Redefine n-step returns (update targets) in terms of estimated action values:

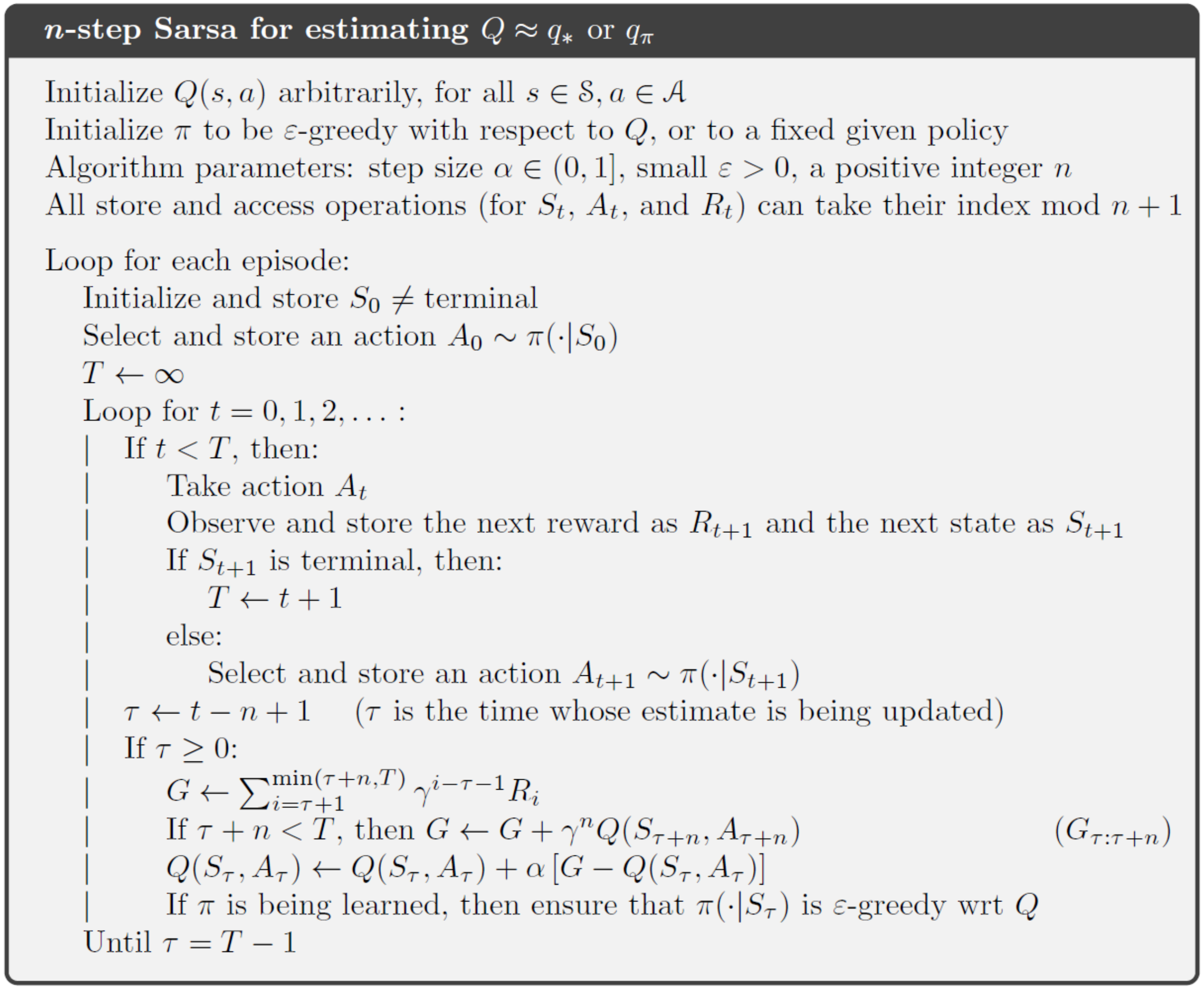
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} \mathrm{R}_{\mathrm{t+n}} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \qquad n \geq 1, 0 \leq t < T - n$$

  with $G_{t:t+n} = G_t$ if $t + n \geq T$
- Then, the update algorithm is

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \qquad 0 \leq t < T$$

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 $a \neq A_t$

n-step Sarsa



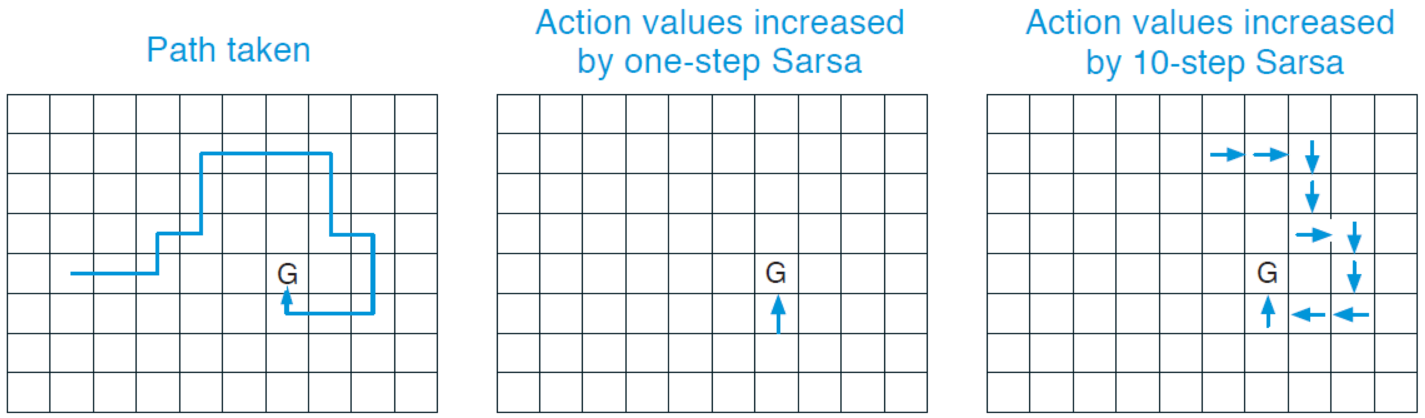
n-step Sarsa for estimating $Q \approx q_*$ or q_π

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

Sutton and Barto, Reinforcement Learning, 2018

Example: Gridworld


Path taken
Action values increased by one-step Sarsa
Action values increased by 10-step Sarsa
G
G
G

Sutton and Barto, Reinforcement Learning, 2018

- Speedup of policy learning due to the use of n-step methods
- G is a location of high reward
- The values were all initially 0, and all rewards were zero except for a positive reward at G
- The one-step method strengthens only the last action of the sequence of actions that led to the high reward
- The n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode

Outline

- ✓ n-step TD Prediction
- ✓ n-step Sarsa
- ➡ ☐ **n-step Off-policy Learning**
- ☐ n-step Tree Backup
- ☐ Conclusion

n-step Off-policy Learning

- Recall that off-policy learning is learning the value function for one policy π, while following another policy b
- Often, π is the greedy policy for the current action-value function estimate, and b is a more exploratory policy, perhaps ϵ-greedy
- In n-step methods, returns are constructed over n steps, so we are interested in the relative probability of just those n actions
- A simple off-policy version of n-step TD is to weight the update for time t by $\rho_{t:t+n-1}$

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

- $\rho_{t:t+n-1}$ is the importance sampling ratio: the relative probabilities under the two policies of taking the n actions from A_t to A_{t+n-1}

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

n-step Off-policy Learning

- If any one of the actions would never be taken by π (i.e., $\pi\ (A_k|S_k) = 0$), then the n-step return should be given zero weight and be totally ignored
- On the other hand, if by chance an action is taken such that π would take 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 (and therefore we want to learn about it) but is selected only rarely by b and thus rarely appears in the data; to make up for this we have to over-weight it when it does occur
- If the two policies are actually the same (the on-policy case) then the importance sampling ratio is always 1

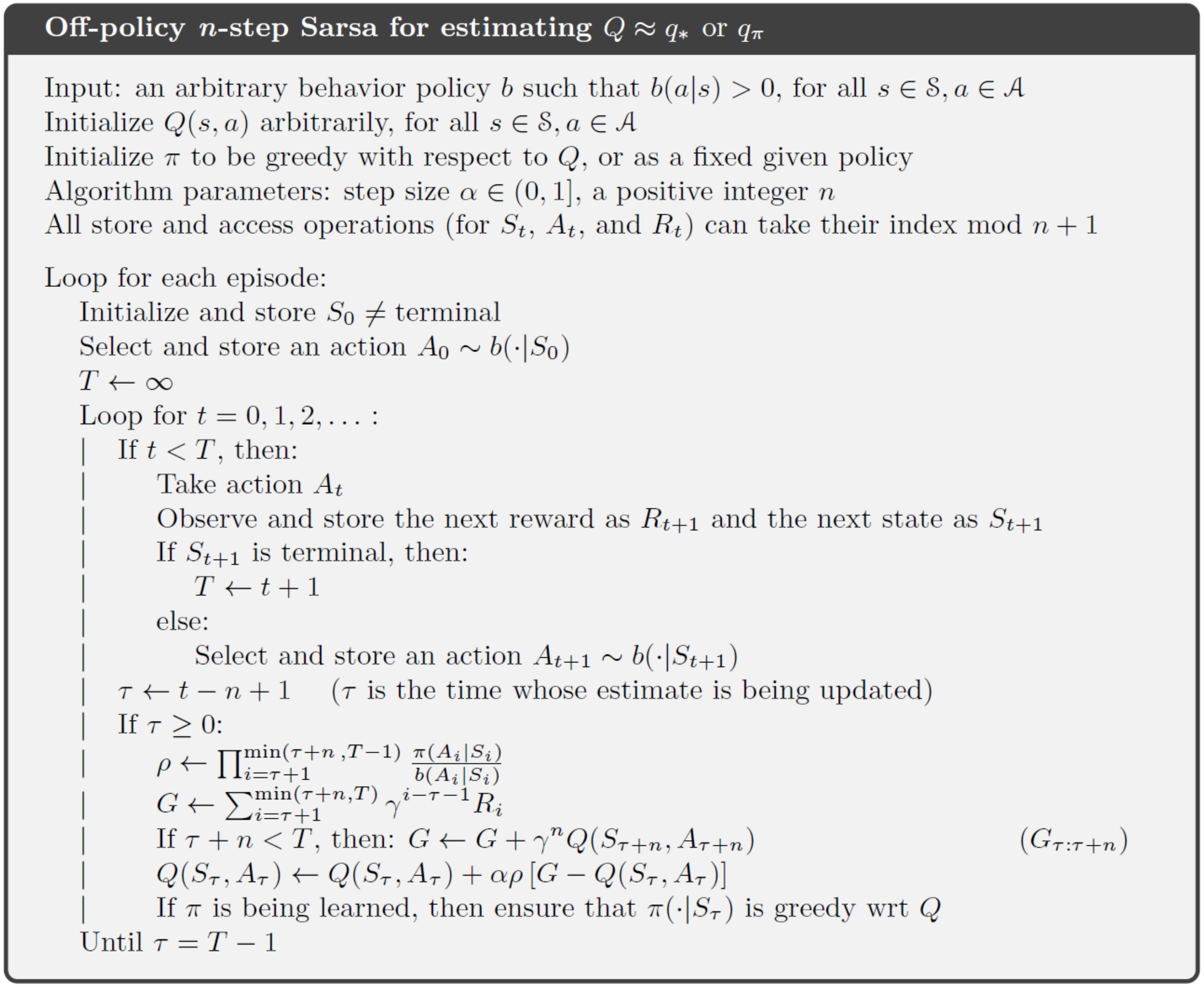
n-step Off-policy Learning

- Similarly, n-step Sarsa update can be completely replaced by a simple off-policy form:

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

  for $0 \leq t < T$
- The importance sampling ratio here starts and ends one step later than for n-step TD, since we are updating a state–action pair
- We do not have to care how likely we were to select the action; now that we have selected it we want to learn fully from what happens, with importance sampling only for subsequent actions

n-step Off-policy Learning



```
Off-policy n-step Sarsa for estimating $Q \approx q_*$ 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 $A_{t+1} \sim b(\cdot|S_{t+1})$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose estimate is being updated)
    |   If $\tau \geq 0$:
    |       $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       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 $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt $Q$
    Until $\tau = T - 1$
```

Sutton and Barto, Reinforcement Learning, 2018

n-step Off-policy Learning

- Off-policy version of n-step Expected Sarsa
  - Use the same update as that of n-step Sarsa except that the importance sampling ratio would have one less factor in it
  - The update equation would use $\rho_{t+1:t+n-1}$ instead of $\rho_{t+1:t+n}$, and of course it would use the Expected Sarsa version of the n-step return
  - This is because in Expected Sarsa all possible actions are taken into account in the last state; the one actually taken has no effect and does not have to be corrected for

Outline

- ✓ n-step TD Prediction
- ✓ n-step Sarsa
- ✓ n-step Off-policy Learning
- ➡ ☐ **n-step Tree Backup**
- ☐ Conclusion

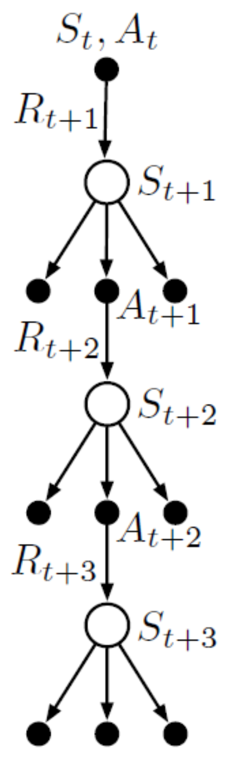
n-step Tree Backup

- Is off-policy learning possible without importance sampling?
- Q-learning and Expected Sarsa do this for the one-step case, but is there a corresponding multi-step algorithm?

- Answer: tree-backup algorithm

n-step Tree Backup

- The figure shows three sample states and rewards, and two sample actions; these are the random variables representing the events occurring after the initial state–action pair S_t, A_t
- Hanging off to the sides of each state are the actions that were not selected (for the last state, all the actions are considered to have not (yet) been selected)
- Because we have no sample data for the unselected actions, we bootstrap and use the estimates of their values in forming the target for the update
- So far we have always updated the estimated value of the node at the top of the diagram toward a target combining the rewards along the way (appropriately discounted) and the estimated values of the nodes at the bottom
- In the tree-backup update, the target includes all these things plus the estimated values of the dangling action nodes hanging off the sides, at all levels

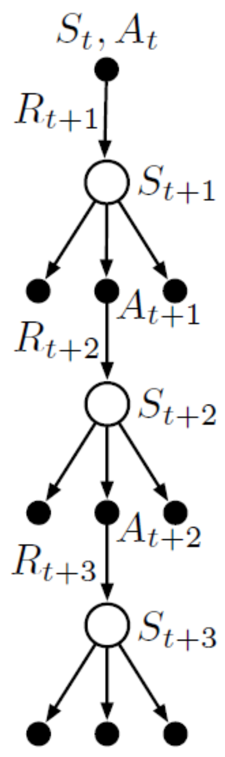


the 3-step tree-backup update

Sutton and Barto, Reinforcement Learning, 2018

n-step Tree Backup

- The update is from the estimated action values of the leaf nodes of the tree; the action nodes in the interior, corresponding to the actual actions taken, do not participate
- Each leaf node contributes to the target with a weight proportional to its probability of occurring under the target policy π
- Each first-level action a contributes with a weight of $\pi(a|S_{t+1})$, except that the action actually taken, A_{t+1}, does not contribute at all; its probability $\pi(A_{t+1}|S_{t+1})$ is used to weight all the second-level action values
- Each non-selected second-level action a' contributes with weight $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$; each third-level action a'' contributes with weight $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a''|S_{t+3})$, and so on



the 3-step tree-backup update

Sutton and Barto, Reinforcement Learning, 2018

n-step Tree Backup

- One-step tree backup return (target): the same as that of Expected Sarsa

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

  for $t < T - 1$

- Two-step tree backup return:

$$\begin{aligned} G_{t:t+2} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ &\qquad + \gamma\pi(A_{t+1}|S_{t+1})\left(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a)\right) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma\pi\left(A_{t+1}\middle|S_{t_1}\right) G_{t+1:t+2}, \end{aligned}$$

  for $t < T - 2$

n-step Tree Backup

- General recursive definition of the tree-backup n-step return:

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

  for $t < T - 1, n \geq 2$

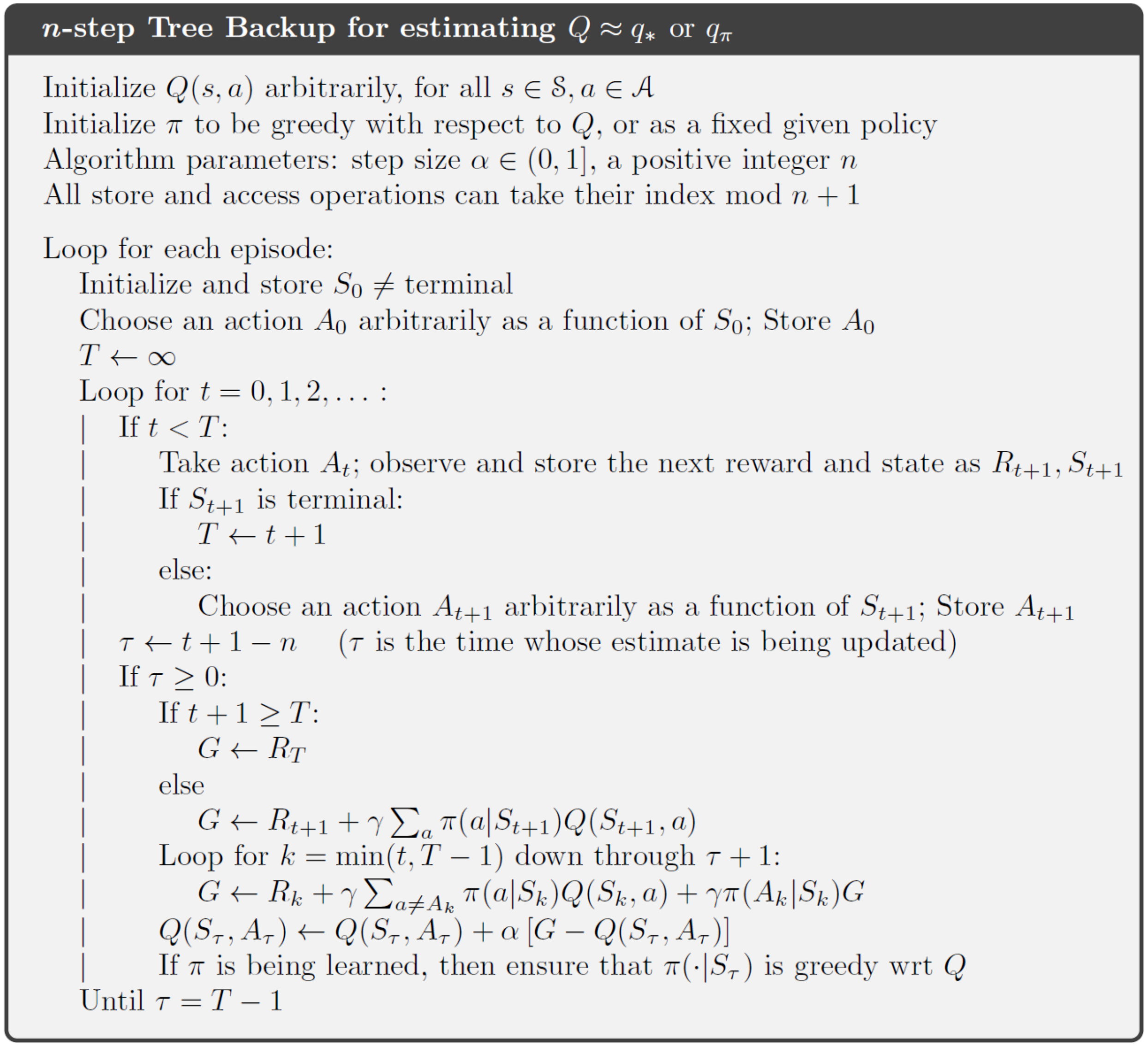
- This target is used with the usual action-value update rule from n-step Sarsa:

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

- for $0 \leq t < T$, while the values of all other state-action pairs remain unchanged: $Q_{t+n}(s,a) = Q_{t+n-1}(s,a)$, for all s, a such that $s \neq S_t$ or $a \neq A_t$

n-step Tree Backup

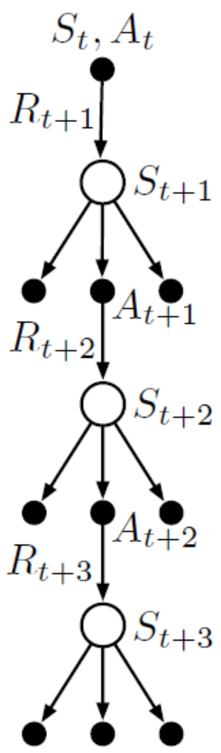
Let n=3, T=6
t = 0 1 2 3 4 5 6
τ = 0 1 2 3 4 5



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 $\pi$ 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 $\tau = T-1$
```



the 3-step tree-backup update

Sutton and Barto, Reinforcement Learning, 2018

Outline

- ✓ n-step TD Prediction
- ✓ n-step Sarsa
- ✓ n-step Off-policy Learning
- ✓ n-step Tree Backup
- ☐ **Conclusion**

Conclusion

- We discussed a range of temporal-difference learning methods that lie in between the one-step TD methods and the MC methods
- Methods that involve an intermediate amount of bootstrapping are important because they will typically perform better than either extreme

Conclusion

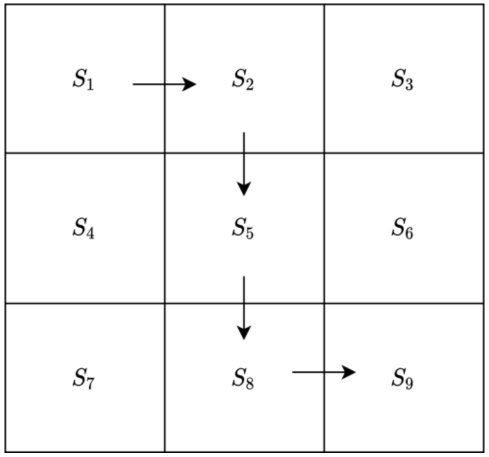
- n-step methods look ahead to the next n rewards, states, and actions
- All n-step methods involve a delay of n time steps before updating, as only then are all the required future events known
- A further drawback is that they involve more computation per time step than previous methods
- Compared 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
- Later, we will see how multi-step TD methods can be implemented with minimal memory and computational complexity using eligibility traces, but there will always be some additional computation beyond one-step methods; such costs can be well worth paying to escape the tyranny of the single time step

Conclusion

- Although n-step methods are more complex than those using eligibility traces, they have the great benefit of being conceptually clear
- Two approaches to off-policy learning in the n-step case
  - Method based on importance sampling: 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
  - Method based on tree-backup updates: the natural extension of Q-learning to the multi-step case with stochastic target policies; it involves no importance sampling

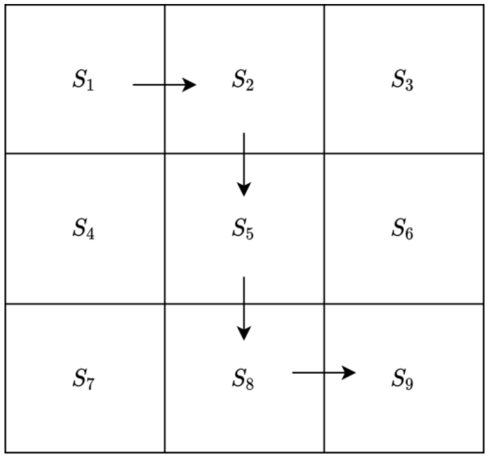
Exercise

- (Question 1)
- Suppose that we have an agent moving around the grid-world, where the available actions are $\{'UP', 'DOWN', 'LEFT', 'RIGHT'\}$. S_1 and S_9 denote the initial state and the terminal state respectively. With the initialized action-value $Q = 0$ for all state action pair and the random $\epsilon - greedy$ policy π, we sampled a trajectory as follows:
- $S_1, RIGHT, -1, S_2, DOWN, -1, S_5, DOWN, -1, S_8, RIGHT, +10, S_9$
- Assuming $\gamma = 0.9$ and $\alpha = 0.5$, use 2-step Sarsa for estimating each of $Q(S_1, RIGHT), Q(S_2, DOWN), Q(S_5, DOWN), and\ Q(S_8, RIGHT)$.



Exercise

- (Answer)
- 2-step Sarsa for estimating Q$\approx q_\pi$
- $S_1, RIGHT, -1, S_2, DOWN, -1, S_5, DOWN, -1, S_8, RIGHT, +10, S_9$
- Update Return: $G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_{t+2-1}(S_{t+2}, A_{t+2})$
- Update Q: $Q_{t+2}(S_t, A_t) \doteq Q_{t+2-1}(S_t, A_t) + \alpha[G_{t:t+2} - Q_{t+2-1}(S_t, A_t)]$
- where, $\gamma = 0.9$, $\alpha = 0.5$, and $Q(s,a) = 0\ \forall(s,a) \in (S,A)$ are given.
-
- $Q(S_1, RIGHT) = \frac{\{-1+0.9(-1)+0.9^2(0)-0\}}{2} = -0.95$
- $Q(S_2, DOWN) = \frac{\{-1+0.9(-1)+0.9^2(0)-0\}}{2} = -0.95$
- $Q(S_5, DOWN) = \frac{\{-1+0.9(10)-0\}}{2} = 4$
- $Q(S_8, RIGHT) = \frac{\{+10+0.9(0)-0\}}{2} = 5$



Questions?