Q-learning and Multi-step TD

Konpat Preechakul
Chulalongkorn University
September 2019

Recap TD and MC

- TD with moving average

$$v^{TD}(s) \leftarrow v(s) + \alpha\big[r + v(s') - v(s)\big]$$

- MC with moving average

$$v^{MC}(s_t) \leftarrow v(s_t) + \alpha\Big[\sum_{\tau=t} r_{\tau+1} - v(s_t)\Big]$$

Recap SARSA

On-policy algorithm

$$q(s_t, a_t) \leftarrow$$
$$q(s_t, a_t) + \alpha\big[r(s_t, a_t) + q(s_{t+1}, a_{t+1}) - q(s_t, a_t)\big]$$

$$q(s_t, a_t) \xleftarrow{\alpha} r(s_t, a_t) + q(s_{t+1}, a_{t+1})$$

- We need (s, a, r, s’, a’) to update
- Hence the name **SARSA** 

Q-learning

Off-policy TD control

- SARSA = on-policy TD control
- SARSA = TD prediction + policy iteration
- If we combine **value iteration** with TD prediction
- We get Q-learning
- Q-learning = TD prediction + value iteration

Q-learning formulation

$$q(s,a) \leftarrow$$

$$q(s,a) + \alpha\Big[r(s,a) + \max_{a'} q(s',a') - q(s,a)\Big]$$

$$q(s,a) \overset{\alpha}{\leftarrow} r(s,a) + \max_{a'} q(s',a')$$

- SARSA with value iteration
- **Needs only (s, a, r, s')**
- SARSA needs (s, a, r, s', a')

Off-policy vs On-policy

On-policy

- Policy can only use experience generated by itself to improve itself

Off-policy

- Policy can use “any” experience to improve itself
- Usually more efficient (less interactions)

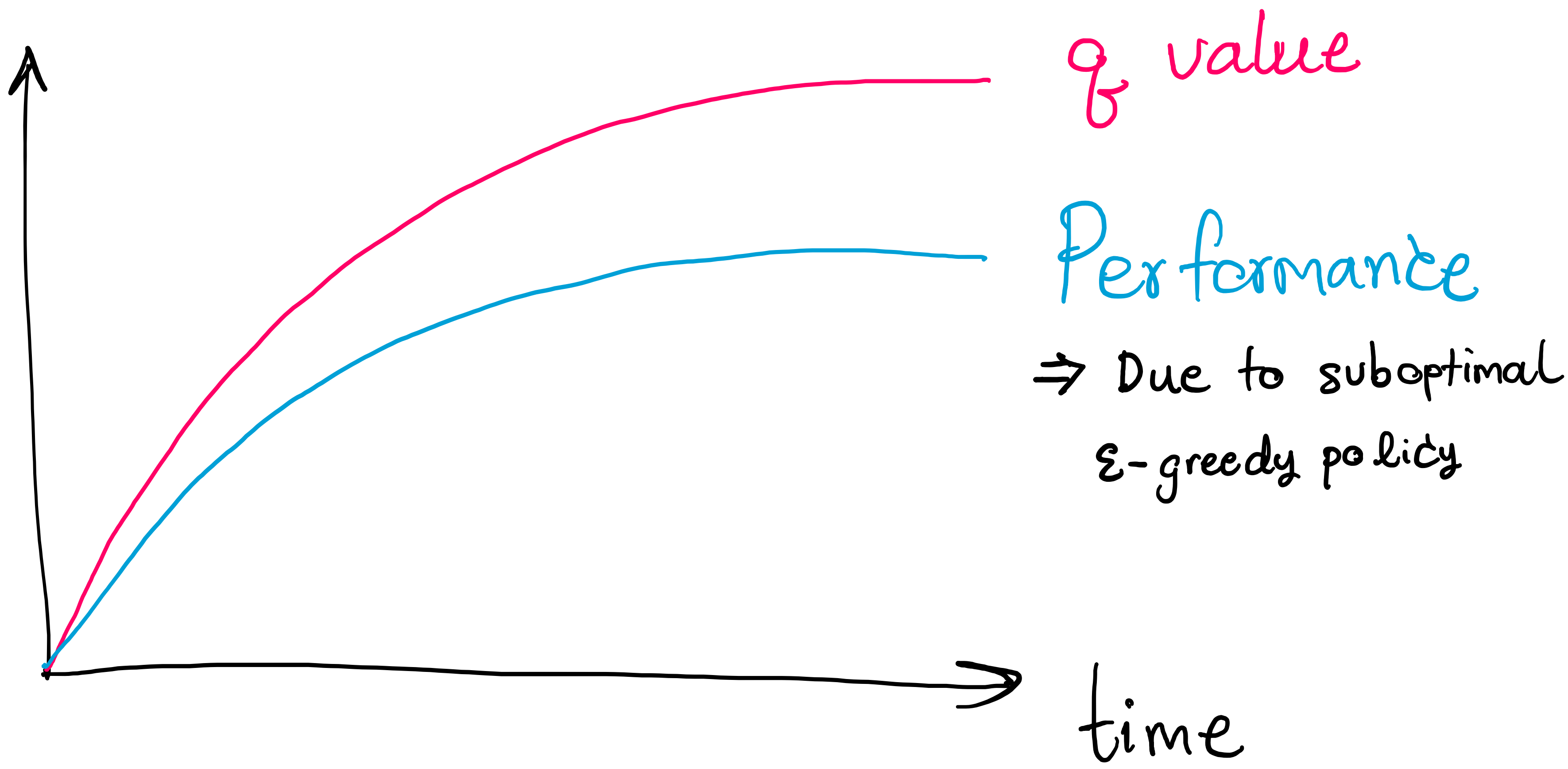
Q-learning with epsilon greedy

- Q-learning's prediction is not aware of epsilon greedy

$$q(s,a) \xleftarrow{\alpha} r(s,a) + \max_{a'} q(s',a')$$

- Its prediction will **NOT** reflect that
- It could lead to risky behavior
- This underlines annealing epsilon

Example

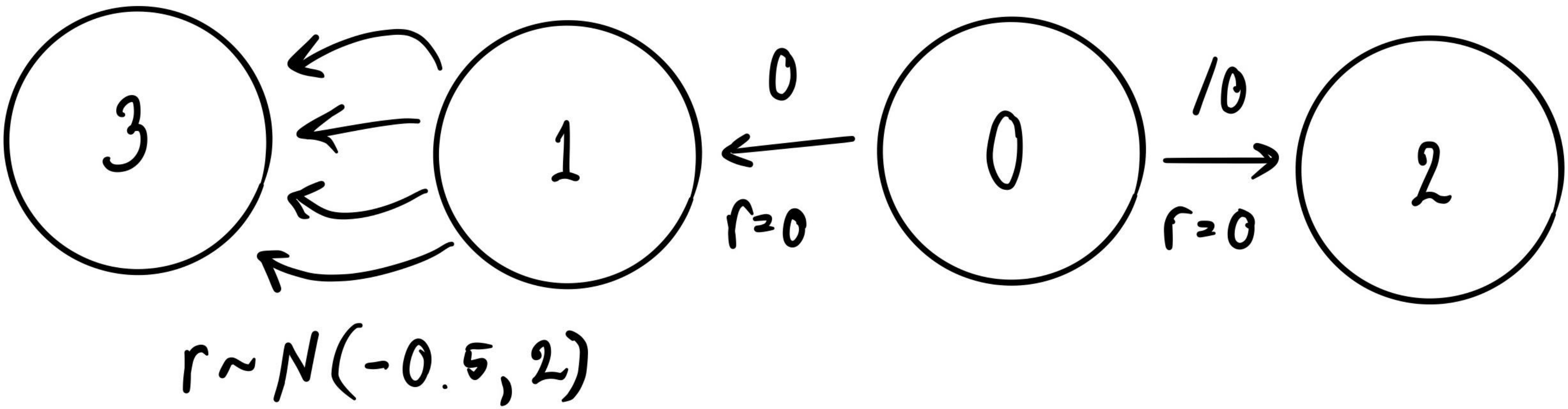


Maximization bias

Maximization bias of Q-learning

- Max-operation needs care when used with sampling
- Max of many random with zero mean is usually larger than mean
- This is a maximization bias
- This happens to SARSA and many algos.

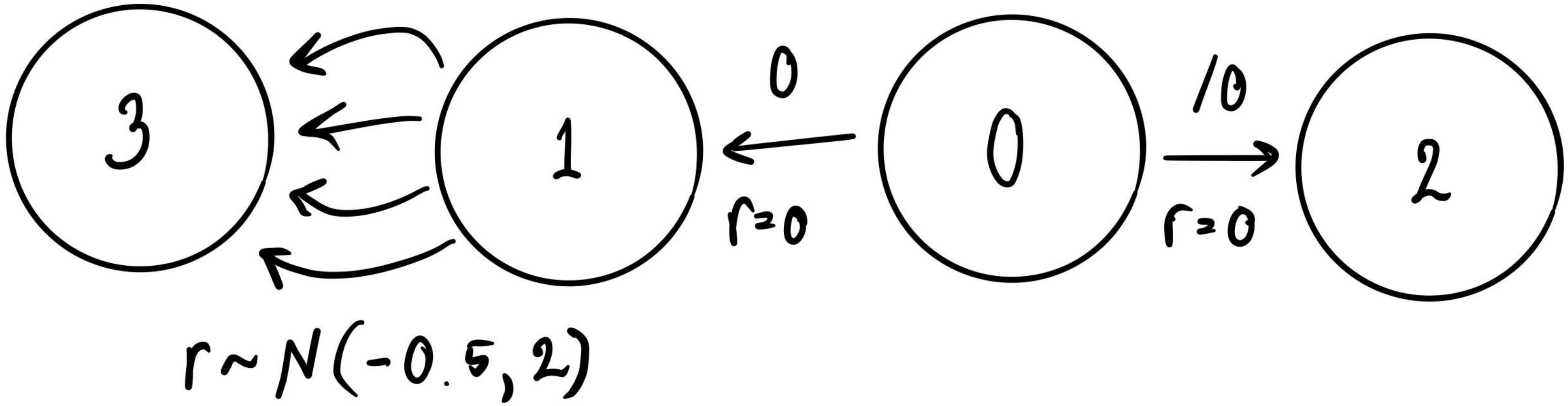
Example



Q-learning is overly optimistic

- We want $\max_a \mathbb{E}\left[q(s,a)\right]$
- Q-learning uses $q(s,a) \approx \mathbb{E}\left[q(s,a)\right]$
- This overestimates the value
- Is it possible mitigate this? How?

Mitigating Maximization Bias



- $\max_a \mathbb{E}\left[q(s,a)\right]$ overestimates because we have stochastic q (due to luck)
- We can reduce this by maintaining 2 **q** functions (don't share exp.)
- If both agree on the max, it is unlikely due to luck

Double Q-learning

Algorithm 1 Double Q-learning

Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

*Double Q-learning “underestimates”

Hasselt, Hado van. 2010. “Double Q-Learning.” In *NIPS 2010*, 2613–21.

N-step TD

A quest for bias-variance balance

Bias	Variance	Algorithm
Low	High	Monte Carlo
High	Low	Temporal Difference
Med	Med	N-step TD?

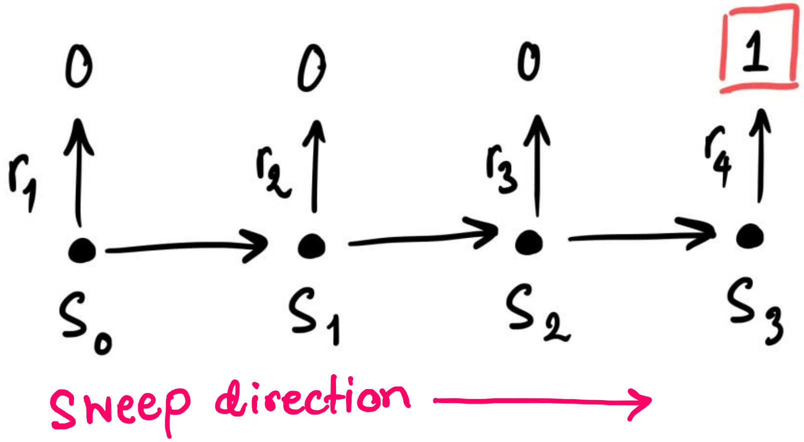
Why we prefer many-steps?

- One step is slow to propagate
- One step has high bias

Why we don't prefer TOO many steps?

- Many-step leads to high variance

Propagating faster example



State	Reward	Next V	R + V'	V
0	0			0
1	0			0
2	0			0
3	1	-		0

From 1 step to Monte Carlo

One-step return

$$g_{t:t+1} = r_{t+1} + \gamma v(s_{t+1})$$

Bootstrapping

Monte Carlo return

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \gamma^{T-t} r_T$$

N-step return

$$g_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \gamma^n v(s_{t+n})$$

Bootstrapping

- Runs for n steps
- Gets n rewards
- Bootstraps the rest

N-step TD

$$v(s_t) \leftarrow v(s_t) + \alpha \big[g_{t:t+n} - v(s_t) \big]$$

- Using n-step return as a target
- Need to wait for n steps for an update
- **Could we implement it online?**
- Online = do something with **1 reward**

Online 3-step TD

$$v(s_t) \leftarrow v(s_t) + \alpha\big[g_{t:t+3} - v(s_t)\big]$$

$$g_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v(s_{t+n})$$

$$\left.\begin{aligned}
\tilde{\delta}_{t+1} &= r_{t+1} + v(s_{t+1}) - v(s_t) \\
\tilde{\delta}_{t+2} &= r_{t+2} + v(s_{t+2}) - v(s_{t+1}) \\
\tilde{\delta}_{t+3} &= r_{t+3} + v(s_{t+3}) - v(s_{t+2})
\end{aligned}\right\} \text{ignore discount}$$

$$\tilde{\delta}_{t+1} + \tilde{\delta}_{t+2} + \tilde{\delta}_{t+3} = r_{t+1} + r_{t+2} + r_{t+3} + v(s_{t+3})$$

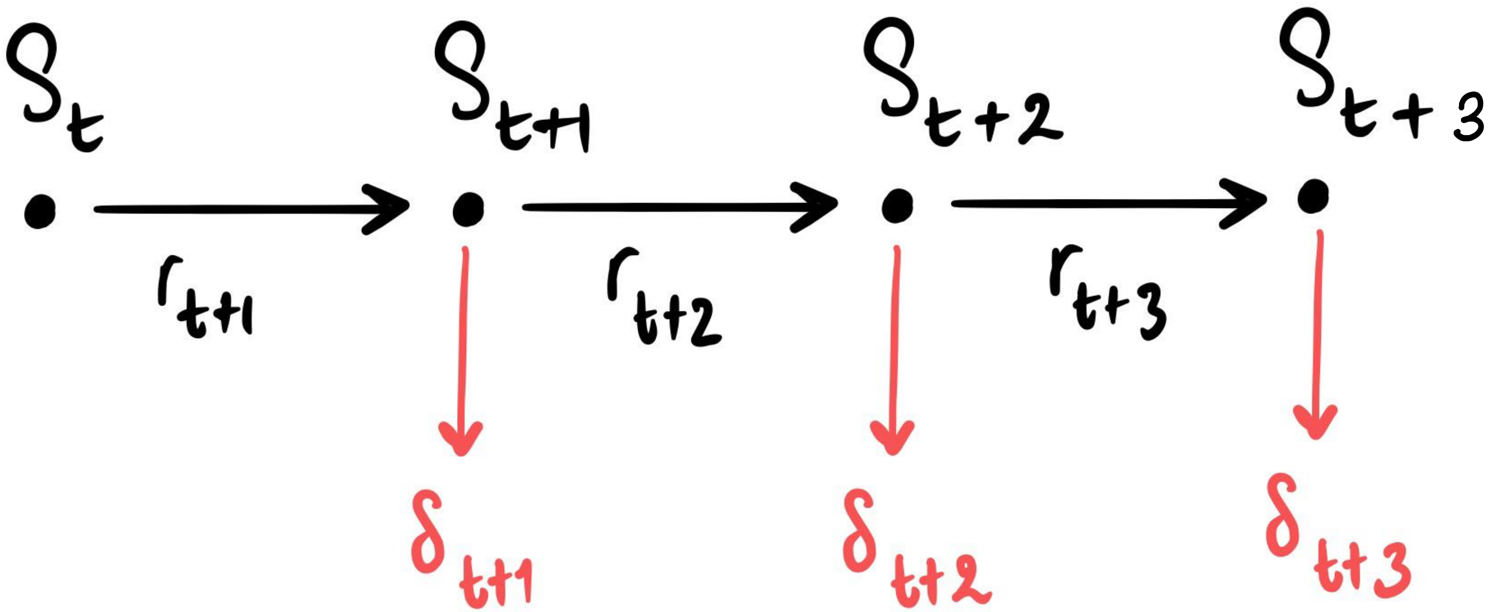
Online 3-step TD

$$\delta_{t+1} = r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$$
$$\gamma\delta_{t+2} = \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - \gamma v(s_{t+1})$$
$$\gamma^2\delta_{t+3} = \gamma^2 r_{t+3} + \gamma^3 v(s_{t+3}) - \gamma^2 v(s_{t+2})$$

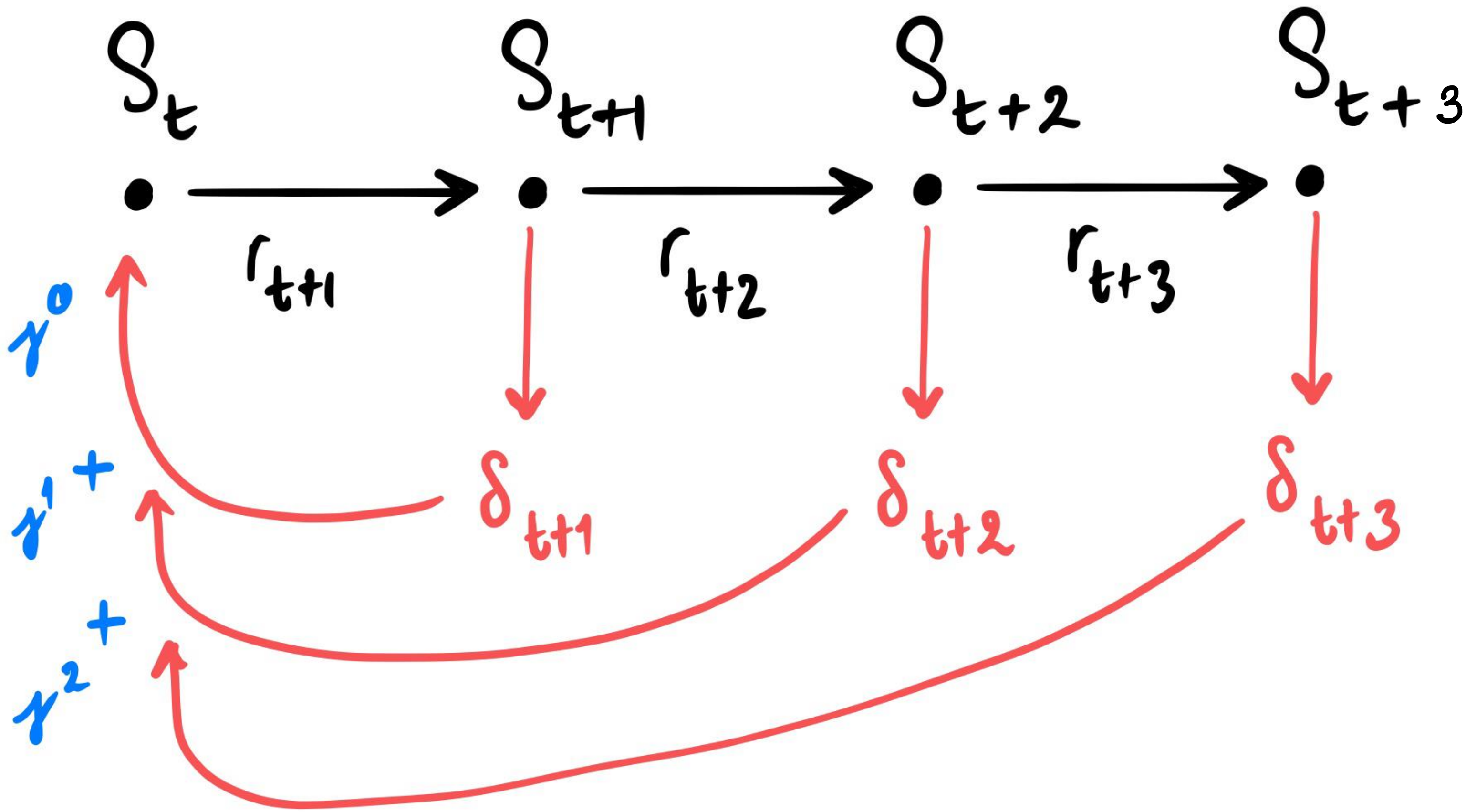
$$\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3}$$
$$= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v(s_{t+n})$$

$$\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3} = g_{t:t+3}$$

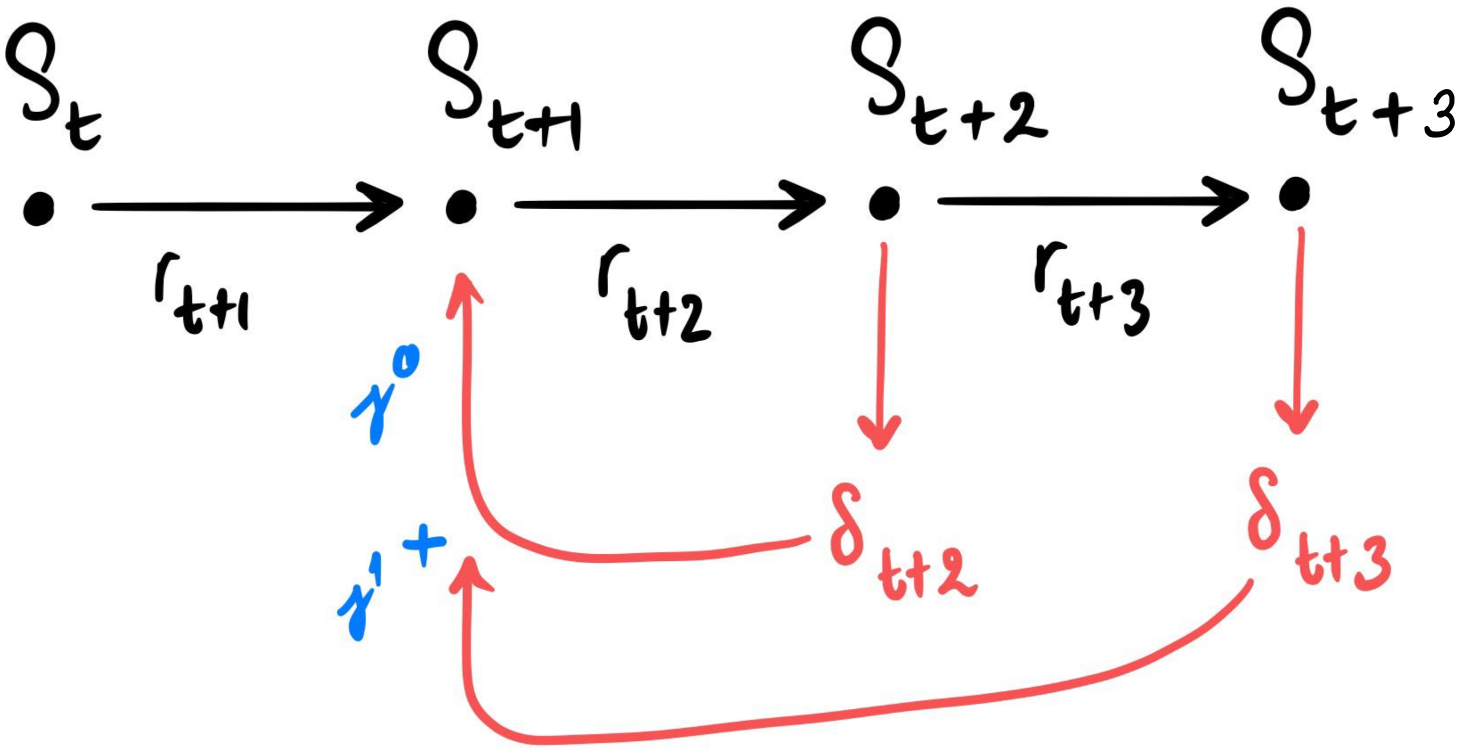
Example



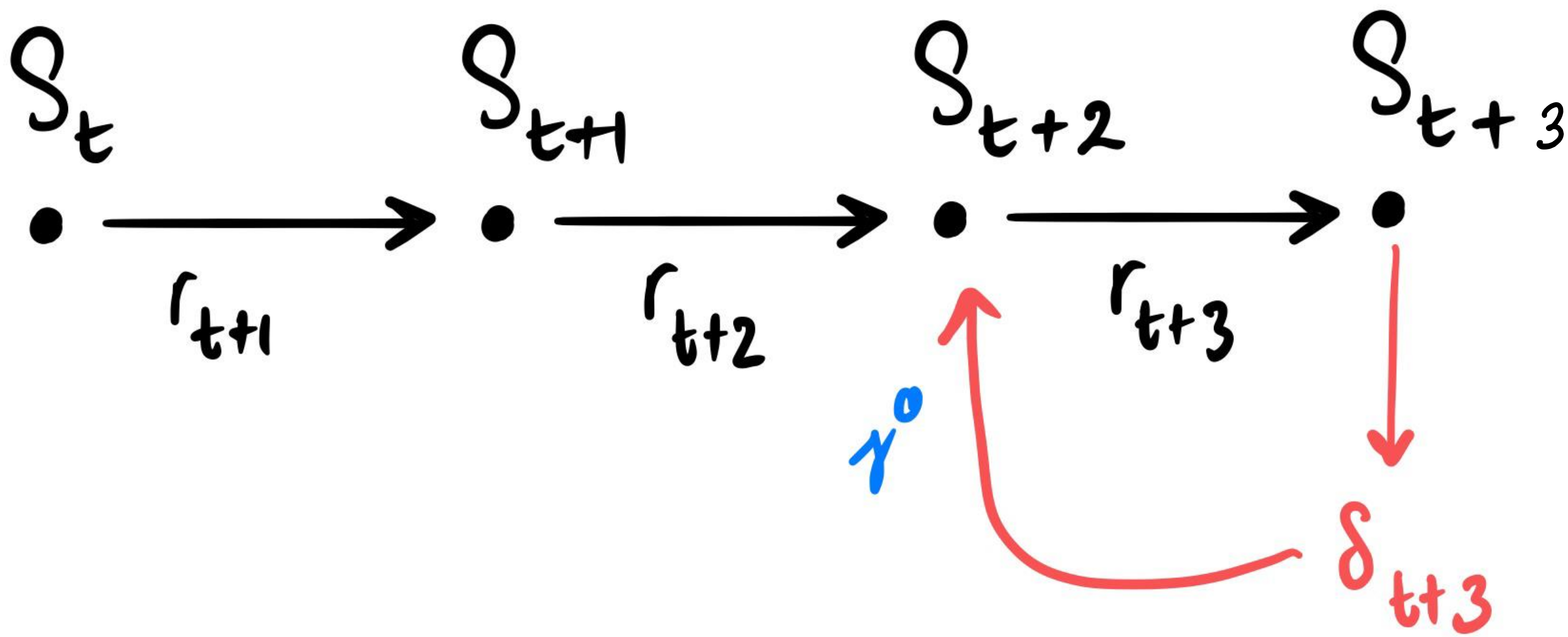
Properly discounted



Properly discounted

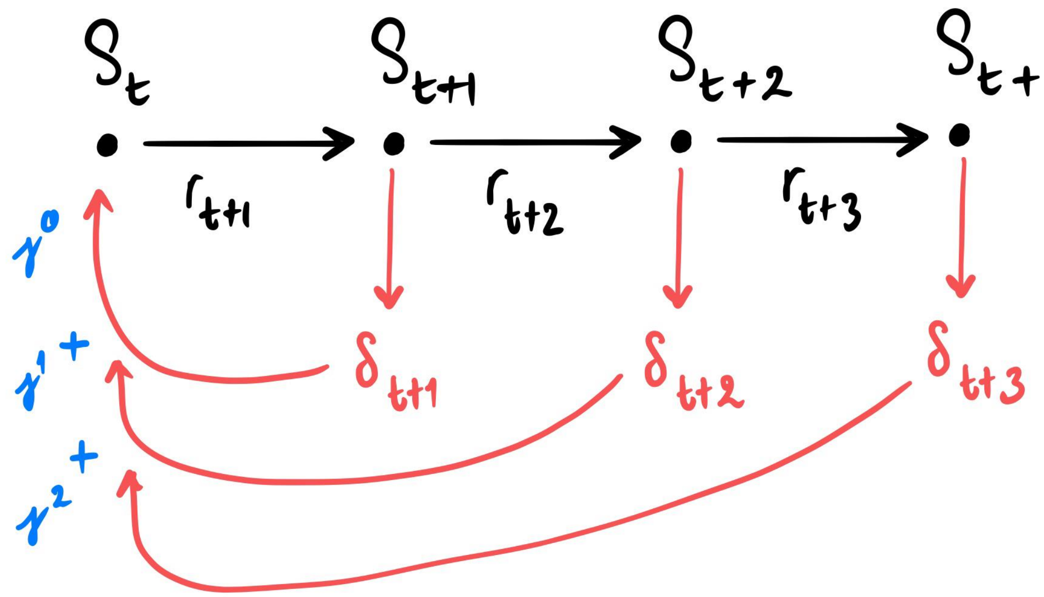


Properly discounted

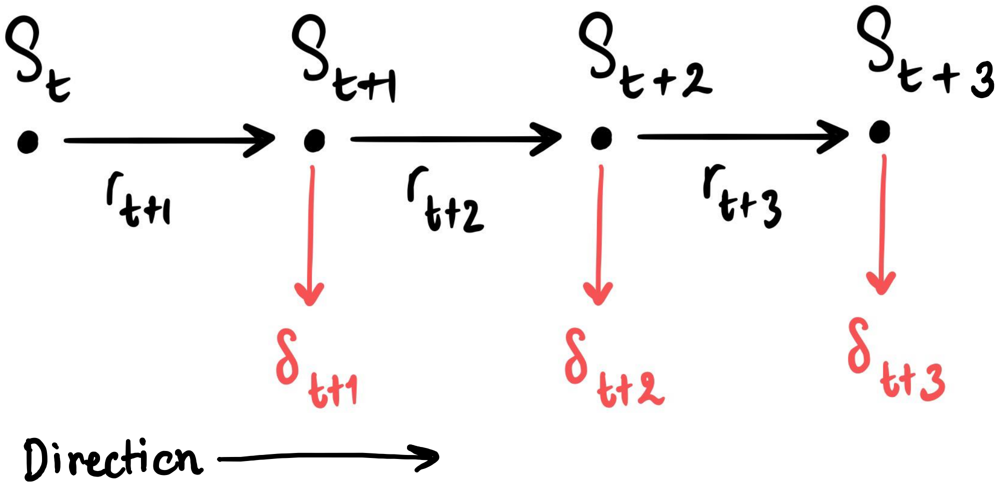


How to keep track of the discount?

- We need some kind of memory
- Called “Eligibility trace”

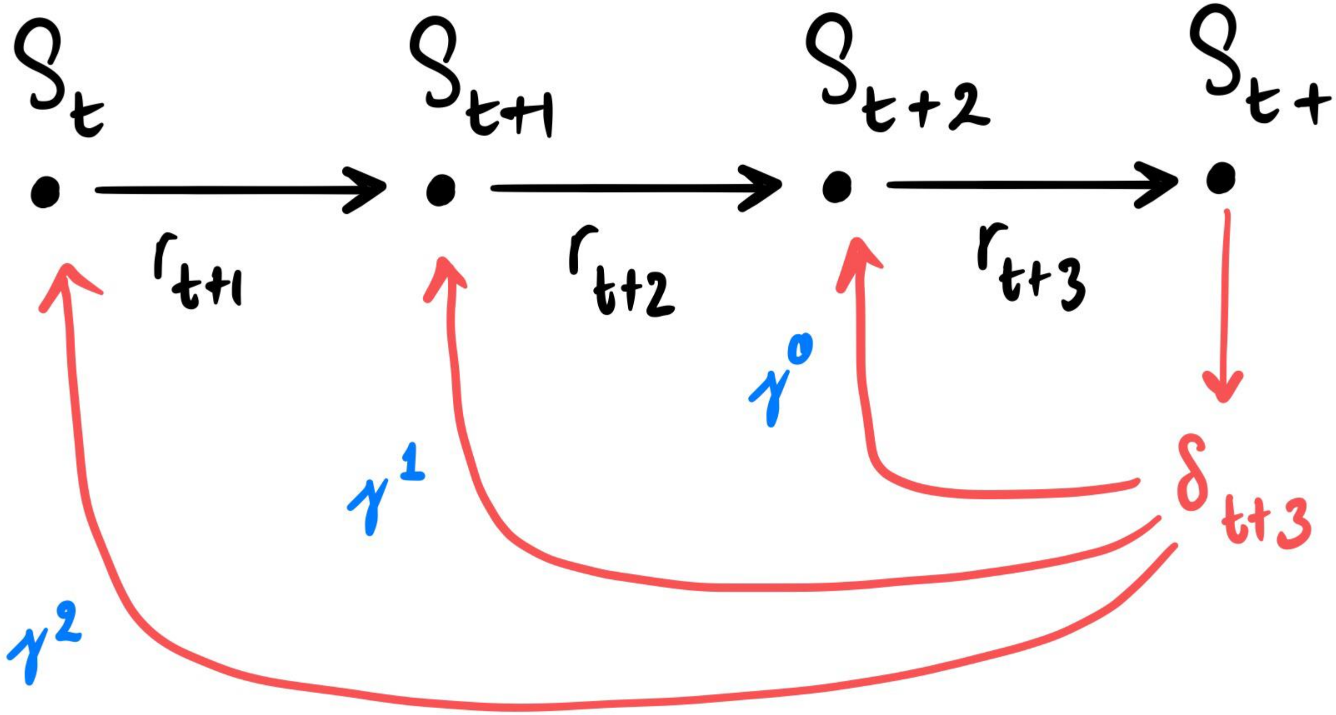


Eligibility keeps track of discount



State	t	t+1	t+2	t+3	t+4
t	0				
t+1	0				
t+2	0				
t+3	0				

Delta updates to previous states



Online N-step TD Algorithm

for until v is stable **do**
  collect experience (s, a, r, s') using π
  $\delta \leftarrow r + v(s') - v(s)$
  $e(s) \leftarrow e(s) + 1$
  for s in S **do**
    $v(s) \leftarrow v(s) + \alpha e(s)\delta$
    $e(s) \leftarrow \gamma e(s)$
    if $e(s) \leq \gamma^n$ **then**
      $e(s) \leftarrow 0$
    end if
  end for
end for

TD(lambda)

If we want something even more “medium”

Lambda return

$$g_\lambda = (1-\lambda)\left[g_{t:t+1} + \lambda g_{t:t+2} + \lambda^2 g_{t:t+3} + \dots\right] + \lambda^{T-t} g_t$$

$$\begin{aligned}
g_\lambda =& (1-\lambda)\lambda^0\left[r_{t+1} + \gamma v(s_{t+1})\right] + \\
& (1-\lambda)\lambda^1\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2})\right] + \\
& (1-\lambda)\lambda^2\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v(s_{t+2})\right] + \\
& \dots + \\
& \underbrace{\lambda^{T-t}}_{\text{Weight}}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots\right]
\end{aligned}$$

$\Sigma = 1$

Lambda return

- Lambda is “share decay rate”
- Each n-step return has its own share
- Lambda => 0 favors small-step return
- Lambda => 1 favors large-step return

Bias	Variance	Lambda
High	Low	Smaller
Low	High	Larger

TD(lambda)

$$v(s_t) \leftarrow v(s_t) + \alpha\left[g_\lambda - v(s_t)\right]$$

$$g_\lambda = (1-\lambda)\left[g_{t:t+1} + \lambda g_{t:t+2} + \lambda^2 g_{t:t+3} + \ldots\right] + \lambda^{T-t} g_t$$

- Naïve implementation needs to wait for a whole episode
- Do we have an online implementation?

Tabular TD Lambda

$$v(s_t) \leftarrow v(s_t) + \alpha\big[g_\lambda - v(s_t)\big]$$

$$\begin{aligned} g_\lambda =&(1-\lambda)\lambda^0\big[r_{t+1} + \gamma v(s_{t+1})\big] + \\ &(1-\lambda)\lambda^1\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2})\big] + \\ &(1-\lambda)\lambda^2\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v(s_{t+2})\big] + \\ &\cdots + \\ &\lambda^{T-t}\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\big] \end{aligned}$$

Tabular TD Lambda

$$v(s_t) \leftarrow v(s_t) + \alpha\left[g_\lambda - v(s_t)\right]$$

$$g_\lambda - v(s_t) =$$

$$(1-\lambda)\lambda^0\left[r_{t+1} + \gamma v(s_{t+1}) - v(s_t)\right] +$$

$$(1-\lambda)\lambda^1\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - v(s_t)\right] +$$

$$(1-\lambda)\lambda^2\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v(s_{t+2}) - v(s_t)\right] +$$

$$\cdots +$$

$$\lambda^{T-t}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\right]$$

Tabular TD Lambda

$$\delta_{t+1} = r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$$

$$\gamma\delta_{t+2} = \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - \gamma v(s_{t+1})$$

$$\gamma^2\delta_{t+3} = \gamma^2 r_{t+3} + \gamma^3 v(s_{t+3}) - \gamma^2 v(s_{t+2})$$

$$\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3}$$

$$= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v(s_{t+n})$$

$$\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3} = g_{t:t+3}$$

Tabular TD Lambda

$$v(s_t) \leftarrow v(s_t) + \alpha\left[g_\lambda - v(s_t)\right]$$

$$g_\lambda - v(s_t) =$$
$$(1-\lambda)\lambda^0\left[\delta_{t+1}\right]+$$
$$(1-\lambda)\lambda^1\left[\delta_{t+1} + \gamma\delta_{t+2}\right]+$$
$$(1-\lambda)\lambda^2\left[\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3}\right]+$$
$$\cdots +$$
$$\lambda^{T-t}\left[\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3} + \dots\right]$$

$$g_\lambda - v(s_t) = ?\delta_{t+1} + ?\gamma\delta_{t+2} + ?\gamma^2\delta_{t+3} + \dots$$

Tabular TD Lambda

$$
\begin{aligned}
&g_\lambda - v(s_t) = \\
&(1-\lambda)\lambda^0 \left[\delta_{t+1}\right] + \\
&(1-\lambda)\lambda^1 \left[\delta_{t+1} + \gamma\delta_{t+2}\right] + \\
&(1-\lambda)\lambda^2 \left[\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3}\right] + \\
&\cdots + \\
&\lambda^{T-t} \left[\delta_{t+1} + \gamma\delta_{t+2} + \gamma^2\delta_{t+3} + \ldots\right]
\end{aligned}
$$

$$1 - (1-\lambda) = \lambda$$

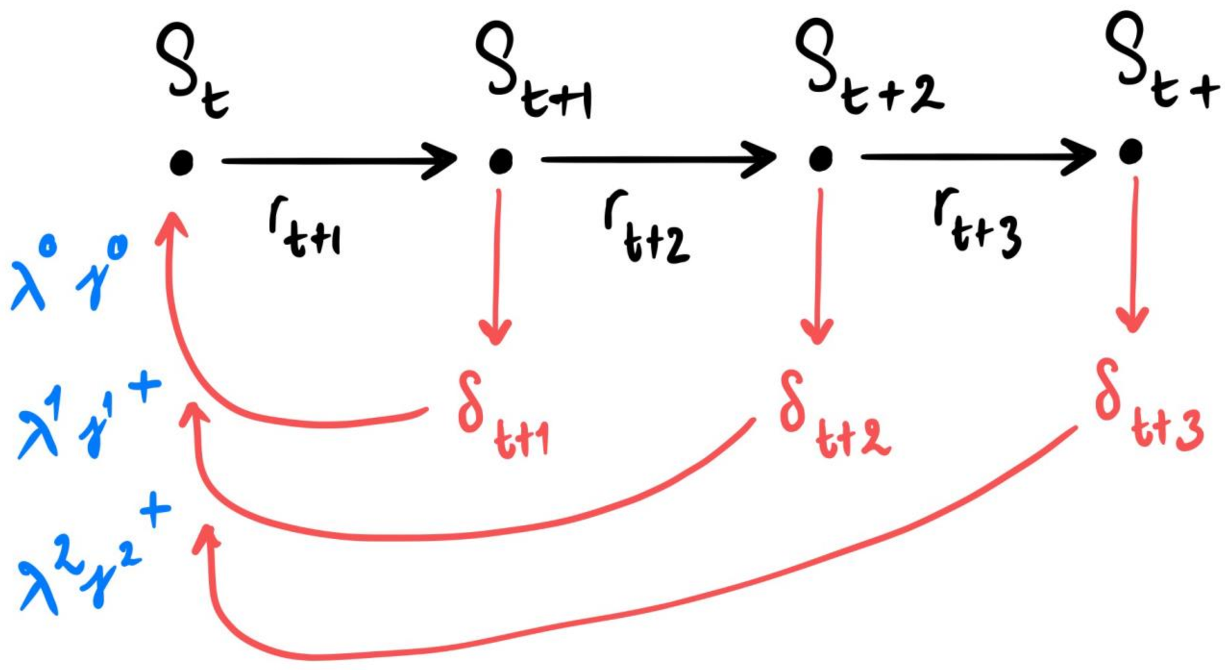
$$1 - (1-\lambda) - (1-\lambda)\lambda = \lambda^2$$

$$1 - (1-\lambda) - (1-\lambda)\lambda - (1-\lambda)\lambda^2 = \lambda^3$$

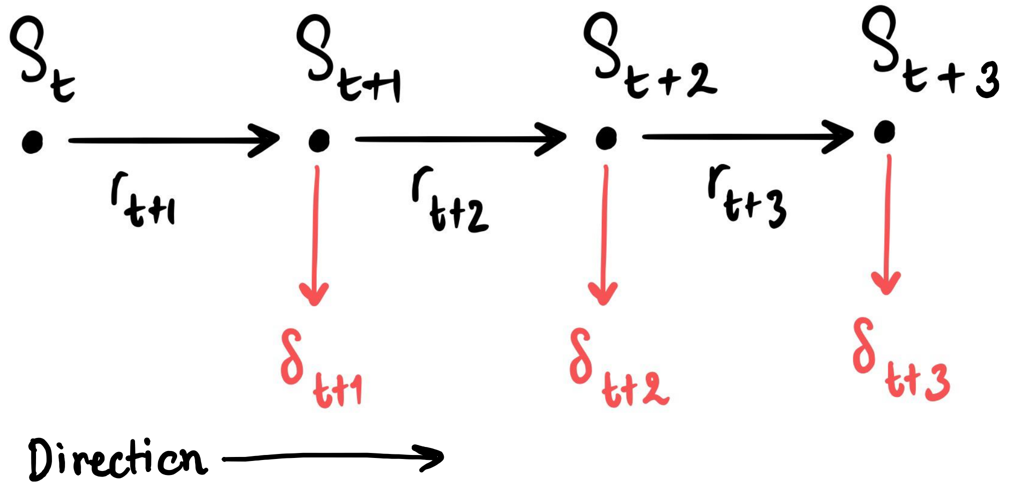
$$g_\lambda - v(s_t) = \delta_{t+1} + \underline{\lambda\gamma}\delta_{t+2} + \underline{\lambda^2\gamma^2}\delta_{t+3} + \ldots$$

Eligibility trace for lambda

- We need to keep track of both "gamma" and "lambda"



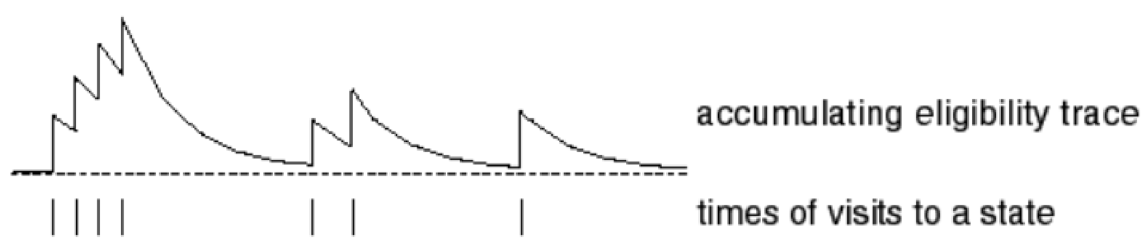
Eligibility trace for lambda



State	t	t+1	t+2	t+3	t+4
t	0				
t+1	0				
t+2	0				
t+3	0				

Online TD(lambda) algorithm

for until v is stable **do**
  collect experience (s, a, r, s') using π
  $\delta \leftarrow r + v(s') - v(s)$
  $e(s) \leftarrow e(s) + 1$
  for s in S **do**
    $v(s) \leftarrow v(s) + \alpha e(s)\delta$
    $e(s) \leftarrow \gamma\lambda e(s)$
  end for
end for



Barto, Andrew G. 2006. “Chapter 7: Eligibility Traces.” 2006.

Online and Offline equivalency

- They are equivalent on paper
- Only applies to “offline” updates
- If you update online, they are no longer the same
- Offline is more “correct”
- More sophisticated kind of trace is needed for better online performance

Online vs Offline performance

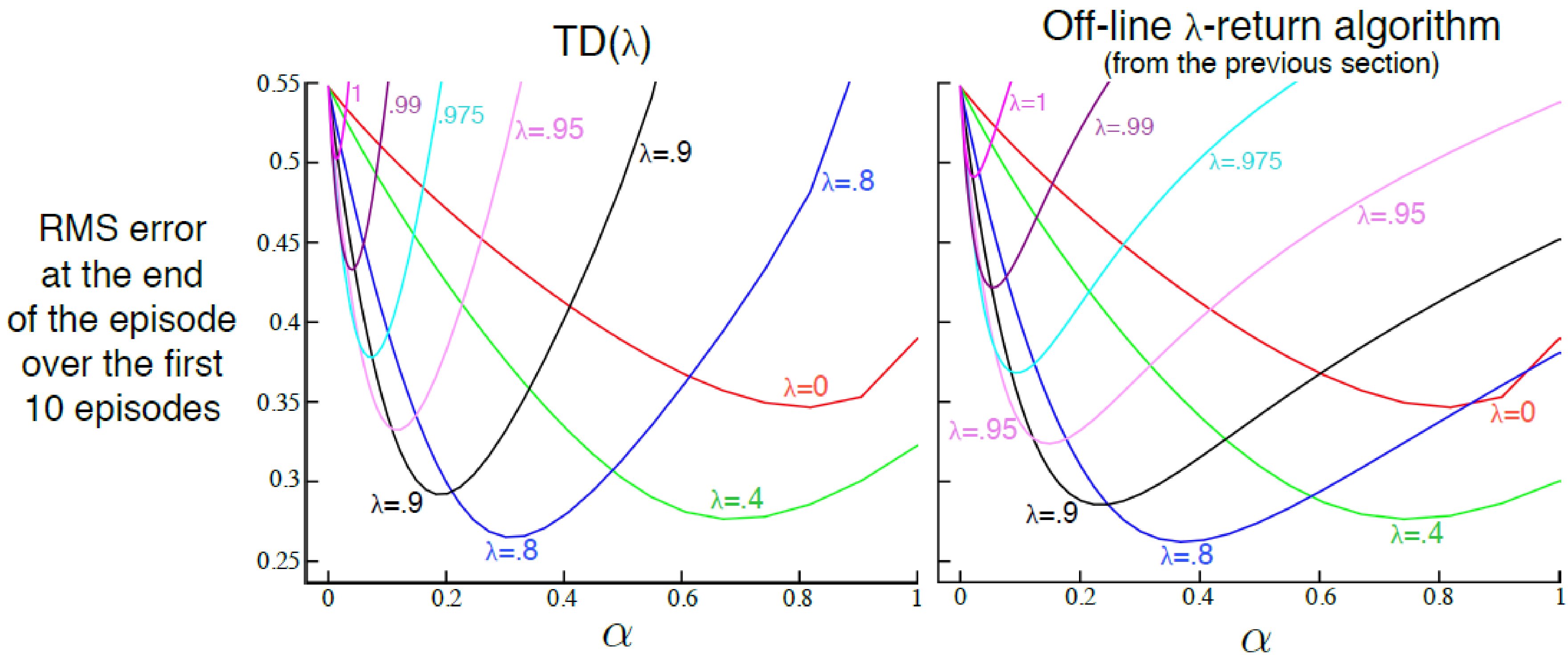


Figure 12.6: 19-state Random walk results (Example 7.1): Performance of TD(λ) alongside that of the offline λ-return algorithm. The two algorithms performed virtually identically at low (less than optimal) α values, but TD(λ) was worse at high α values.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Eligibility trace cons

- We need to update to “all” states
- Inefficient
- In non-tabular case, there is a more efficient eligibility trace

Assignment tour

- Ex 4.1
- Don’t forget to pull