Temporal-difference Learning

Jie-Han Chen
NetDB, National Cheng Kung University
5/15, 2018 @ National Cheng Kung University, Taiwan

Disclaimer

The content and images in this slides were borrowed from:

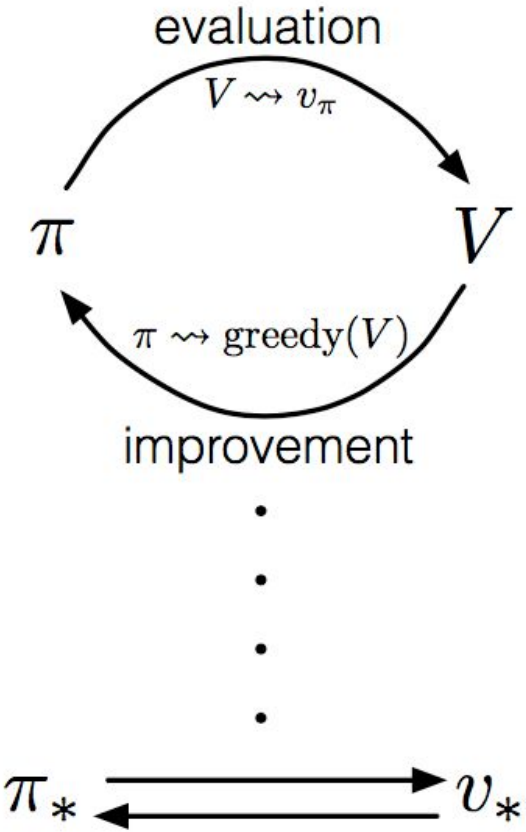
1. Rich Sutton’s textbook
2. David Silver’s Reinforcement Learning class in UCL
3. Künstliche Intelligenz’s slides

Outline

- Recap DP, MC method
- TD learning
- Sarsa, Q-learning
- N-step bootstrapping
- TD (lambda)

Value-based method

Generalized policy iteration (GPI) plays the most important part in reinforcement learning. All we need is to find how to update the value function.



Dynamic Programming

$$V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$$

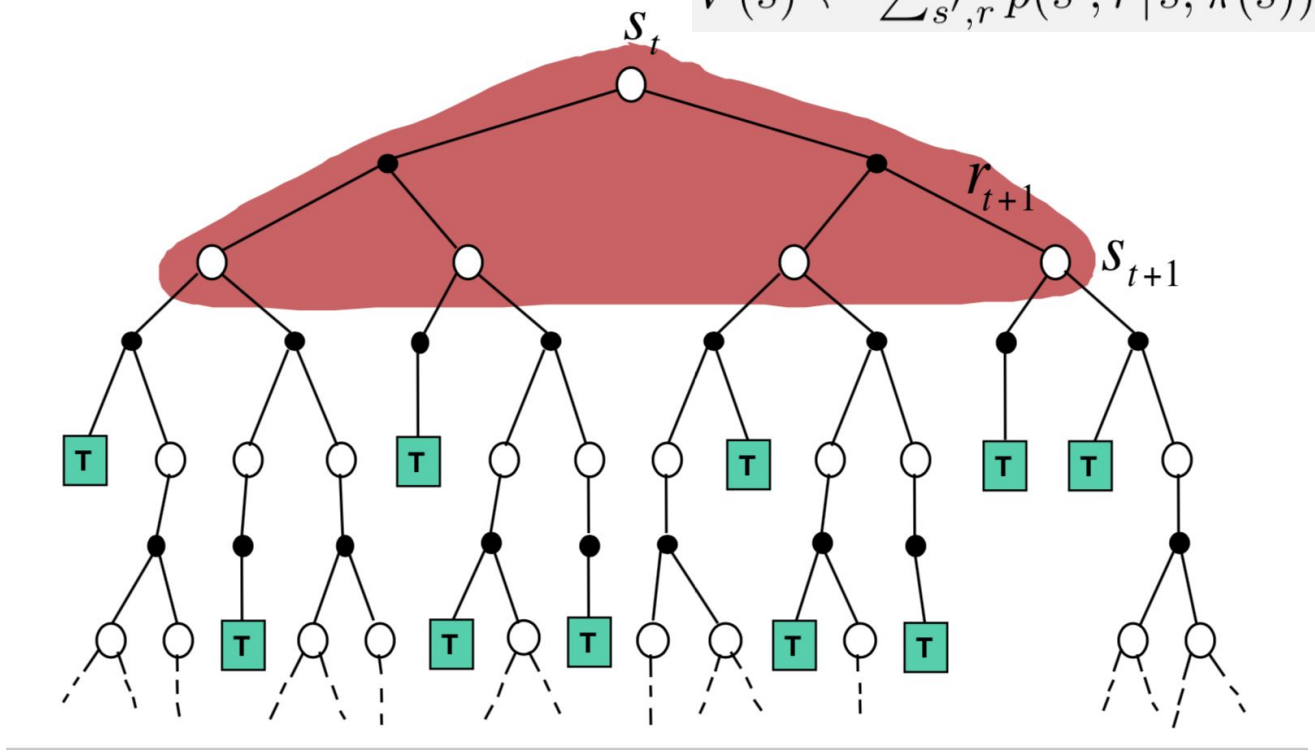


Figure source: David Silver's slides

Monte Carlo Method

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

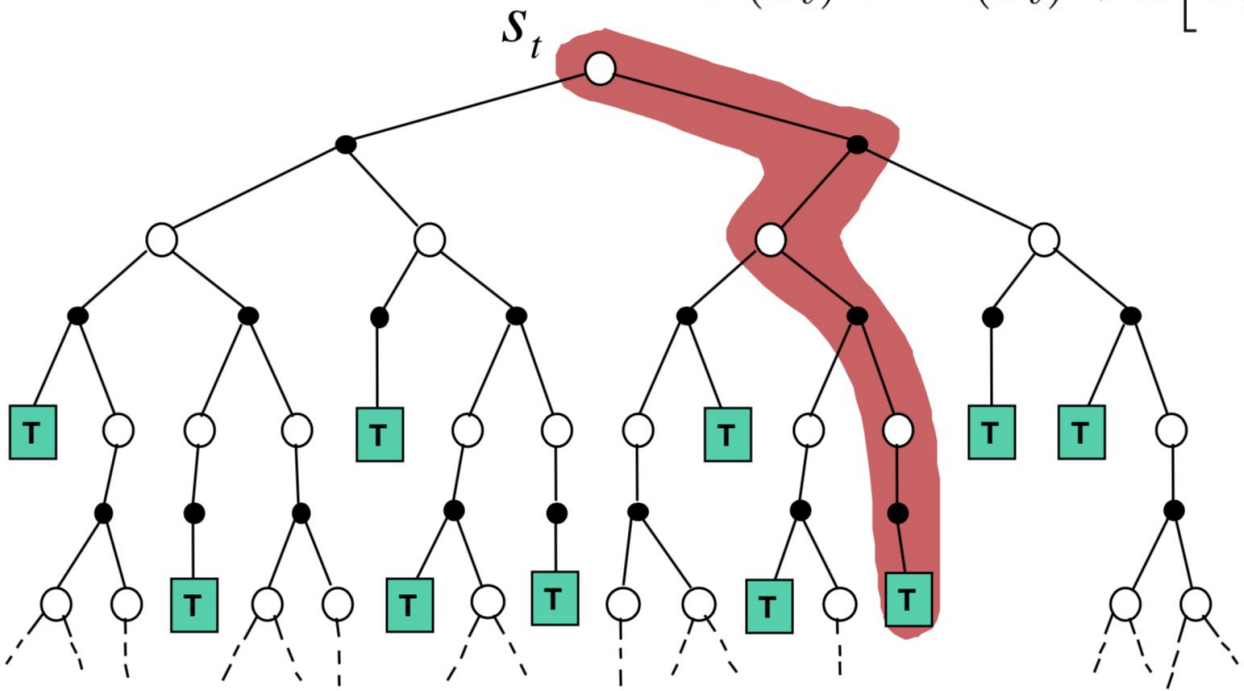


Figure source: David Silver's slides

Dynamic Programming & Monte Carlo method

Dynamic Programming

- update per step, using bootstrapping
- need model
- computation cost

Monte Carlo Method

- update per episode
- model-free
- hard to be applied to continuing task

Can we combine the advantage of Dynamic Programming and Monte Carlo method?

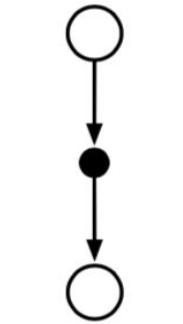
Temporal-difference learning

Temporal-difference learning

Different from MC method, each sample of TD learning is just **a few steps**, not the whole trajectory. TD learning bases its update in part on an existing estimate, so it's also a ***bootstrapping*** method.

TD method is an **policy evaluation method** (without control), which is used to predict the value of fixed policy.

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

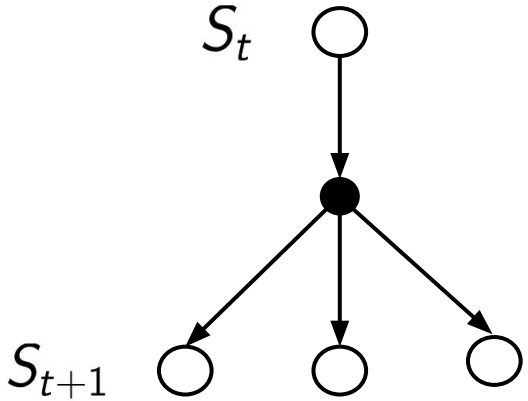


backup diagram of TD(0)

Temporal-difference learning

We want to improve our estimate of V by computing these averages:

$$V_{k+1}(s) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s')+\gamma V_k(s')\right]$$

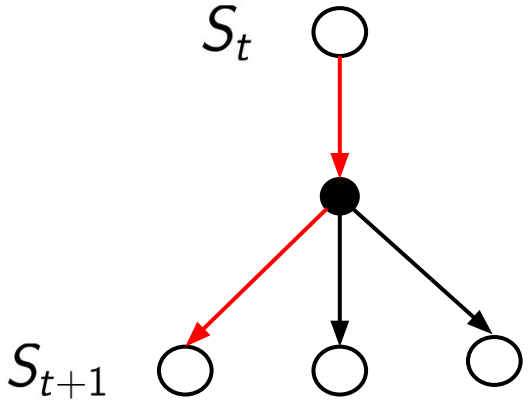


Temporal-difference learning

We want to improve our estimate of V by computing these averages:

$$V_{k+1}(s) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s')+\gamma V_k(s')\right]$$

sample 1: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$



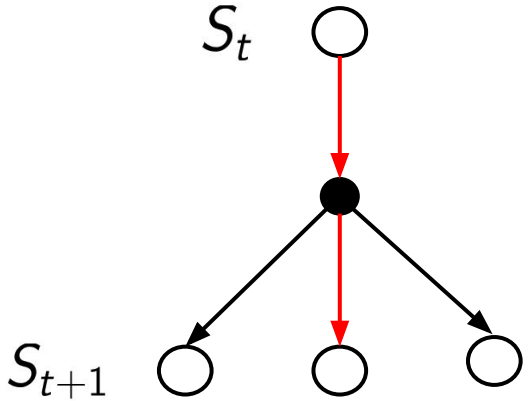
Temporal-difference learning

We want to improve our estimate of V by computing these averages:

$$V_{k+1}(s) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s')+\gamma V_k(s')\right]$$

sample 1: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample 2: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$



Temporal-difference learning

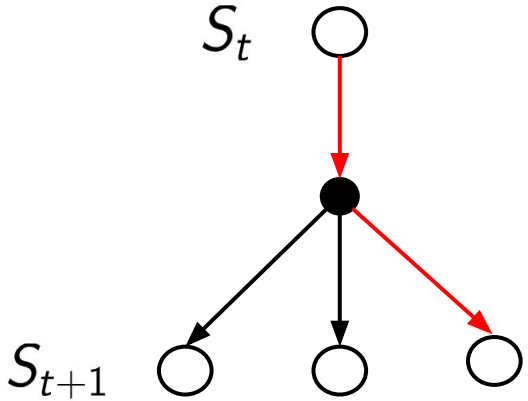
We want to improve our estimate of V by computing these averages:

$$V_{k+1}(s) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s')+\gamma V_k(s')\right]$$

sample 1: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample 2: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample 3: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$



Temporal-difference learning

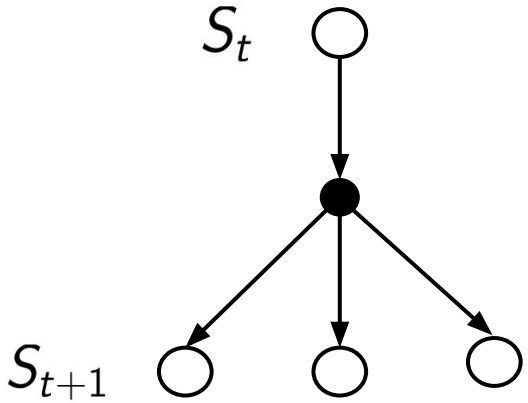
In model-free RL, we use samples to estimate the expectation of future total rewards.

sample 1: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample 2: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

...

sample n: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

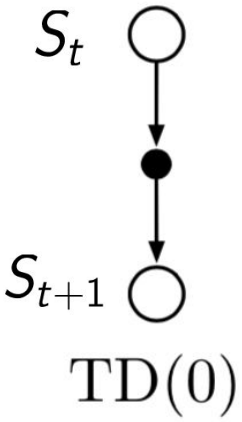


Temporal-difference learning

sample 1: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample 2: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample n: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$



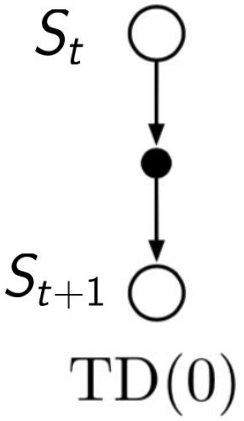
$$V_{k+1}(S_t) = \frac{1}{n} \sum_{i=0} sample_i$$

Temporal-difference learning

sample 1: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample 2: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

sample n: $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$



$$V_{k+1}(S_t) = \frac{1}{n} \sum sample_i$$

But, we cannot rewind time to get sample after sample from St !

Temporal-difference learning

We can use **weighted format** to update the new value function:

$$V(s) = (1 - \alpha)V(s) + (\alpha)sample$$

which is equal to

$$V(s) = V(s) + \alpha(sample - V(s))$$

The α can be a kind of learning rate.

Exponential Moving Average

- The running interpolation update:

$$\overline{x}_n = (1-\alpha) \cdot \overline{x}_{n-1} + \alpha \cdot x_n$$

- Makes recent samples more important:

$$\overline{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

- Forgets about the past, α is its **forget rate**.

Temporal-difference learning

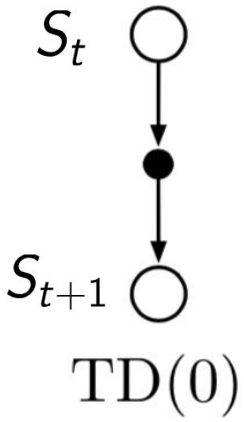
TD update, one-step TD/TD(0):

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

The target of TD method

The quantity in the brackets is a sort of error, called **TD error**.

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$



Temporal-difference learning

- Model-free
- Online learning (fully incremental method)
  - Can be applied to continuing task
- Better convergence time
  - In practice, converge faster than Monte Carlo method

How to choose α ?

Stochastic approximation theory (Robbins-Monro sequence) tells us there are two constraints to make previous exponential moving average converge stably.

1. $$\sum_t \alpha^t = \infty$$

2. $$\sum_t (\alpha^t)^2 < \infty$$

How to choose α ?

Stochastic approximation theory (Robbins-Monro sequence) tells us there are two conditions to make previous exponential moving average converge stably.

1. $\sum_t \alpha^t = \infty$

   P-series could be a choice to satisfy these two conditions. But the learning rate with these conditions will learn slow to converge.

2. $\sum_t (\alpha^t)^2 < \infty$

   The $constant - \alpha$ works well in most case.

Temporal-difference learning with Control

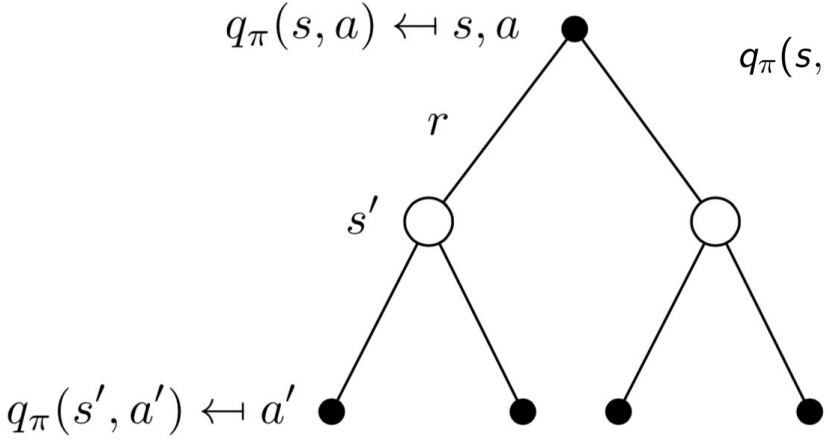
In the previous slides, we introduce TD learning which is used to predict the value function by one-step sample.

Now, we'll introduce two classic methods in TD control:

- Sarsa
- Q-learning

Sarsa

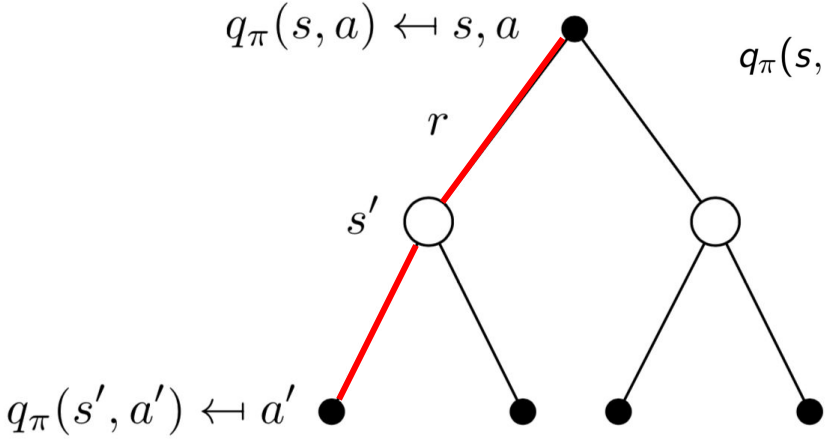
- Inspired by policy iteration
- Replace value function by action-value function



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Sarsa

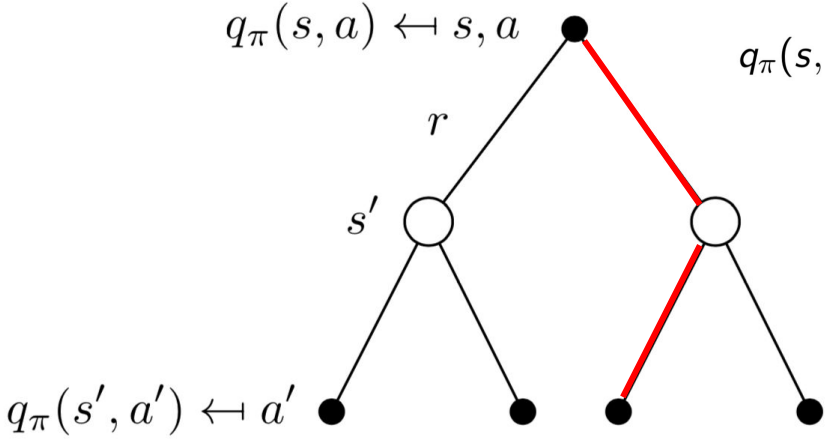
- Inspired by policy iteration
- Replace value function by action-value function



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Sarsa

- Inspired by policy iteration
- Replace value function by action-value function



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Sarsa

- Inspired by policy iteration
- The behavior policy and target policy is same

$$Q_\pi(s,a) \leftarrow \sum_{s'} P(s'|s,\pi(s))\left[R(s,\pi(s),s') + \gamma Q_\pi(s',\pi(s'))\right]$$

Sarsa

- Inspired by policy iteration
- The behavior policy and target policy is same

$$Q_\pi(s,a) \leftarrow \sum_{s'} P(s'|s,\pi(s))\left[R(s,\pi(s),s') + \gamma Q_\pi(s',\pi(s'))\right]$$

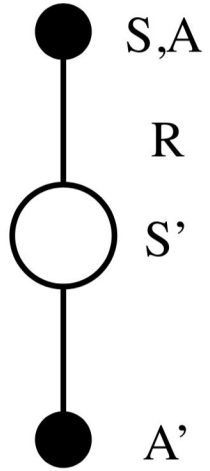
Sarsa

- Inspired by policy iteration
- The behavior policy and target policy is same

$$Q_\pi(s,a) \leftarrow \sum_{s'} P(s'|s,\pi(s))\left[R(s,\pi(s),s') + \gamma Q_\pi(s',\pi(s'))\right]$$

In model-free method, we don't know the transition probability. All we need to do is to use a lot of experience sample to estimate value.

The experience sample in Sarsa is (s, a, r, s', a')



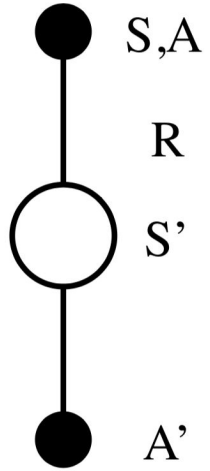
Sarsa

- Inspired by policy iteration

$$Q_\pi(s,a) \leftarrow \sum_{s'} P(s'|s,\pi(s))\left[R(s,\pi(s),s') + \gamma Q_\pi(s',\pi(s'))\right]$$

- Sarsa update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma Q(s',a') - Q(s,a)\right]$$



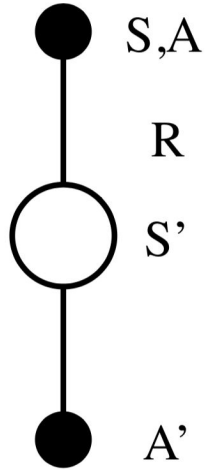
Sarsa

- Inspired by policy iteration

$$Q_\pi(s,a) \leftarrow \sum_{s'} P(s'|s,\pi(s))\left[R(s,\pi(s),s') + \gamma Q_\pi(s',\pi(s'))\right]$$

- Sarsa update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma Q(s',a') - Q(s,a)\right]$$



Sarsa

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 $Q(terminal, \cdot) = 0$

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

Sarsa

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 $Q(terminal, \cdot) = 0$

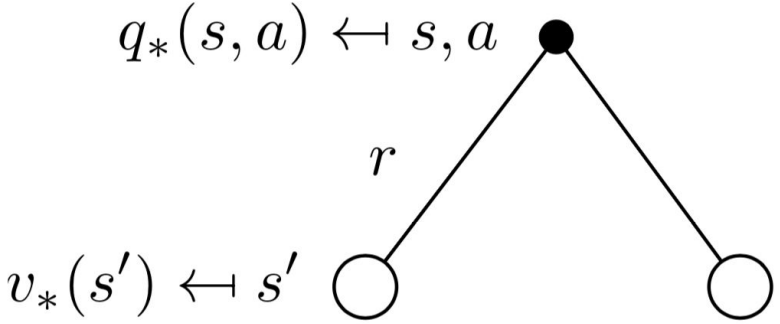
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') -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

On-policy!

Q-learning

- Inspired from value iteration

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

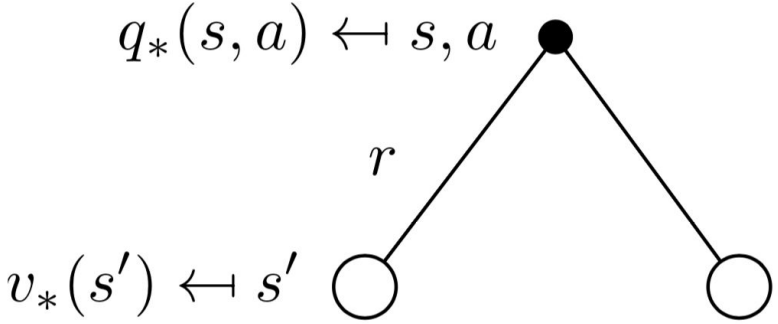


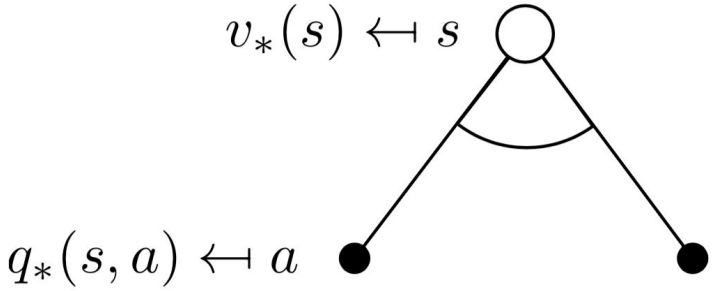
Q-learning

- Inspired from value iteration

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$v_*(s) = \max_a q_*(s,a)$$



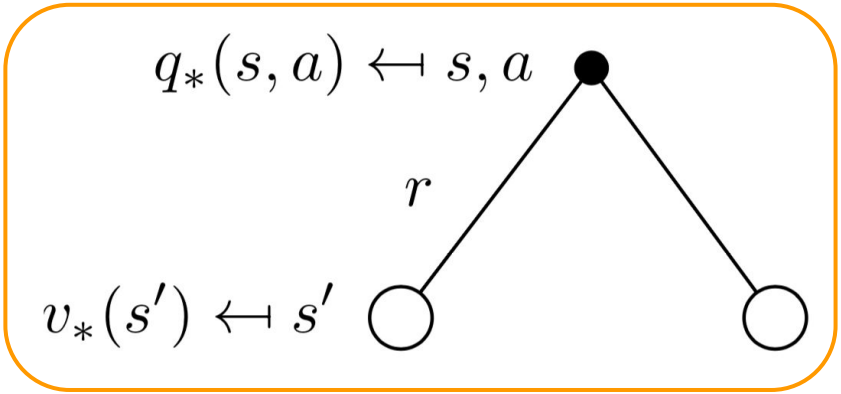


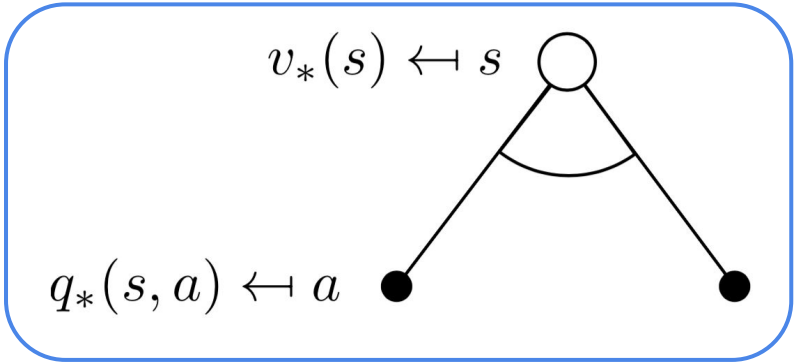
Q-learning

- Inspired from value iteration

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$v_*(s) = \max_a q_*(s,a)$$





Q-learning

- Inspired from value iteration

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$v_*(s) = \max_a q_*(s,a)$$

Q-learning

- Inspired from value iteration

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma max_{a'} Q(s',a')\right]$$

- Q-learning update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Q-learning

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 $Q(\mathit{terminal}, \cdot) = 0$

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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

SARSA V.S. Q-Learning

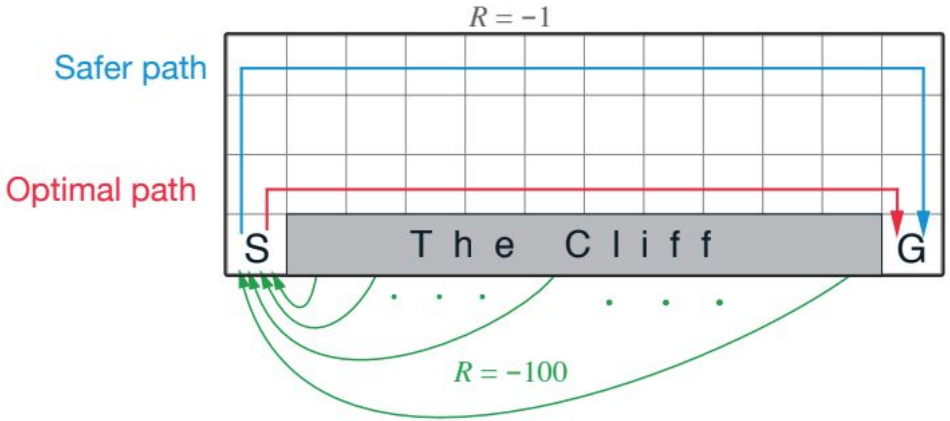
- On-policy: The sample policy is as same as learning policy (target policy)

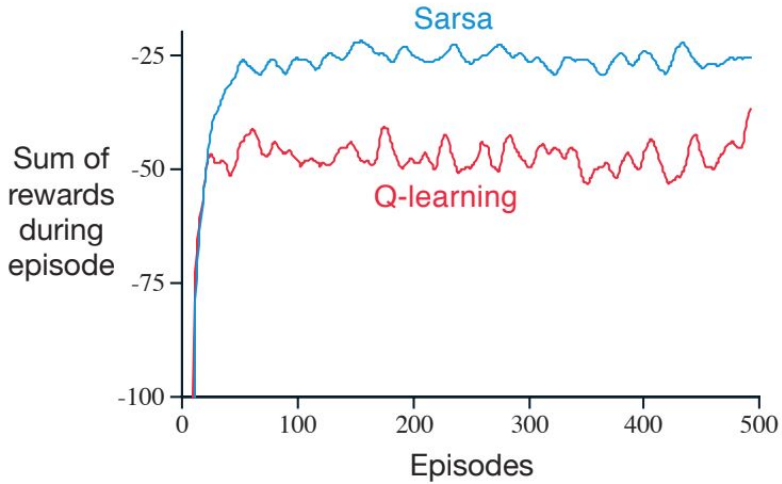
  eg: Sarsa, Policy Gradient

- Off-policy: The sample policy is different from learning policy (target policy)

  eg. Q-learning, Deep Q Network

SARSA V.S. Q-Learning: The Cliff walk

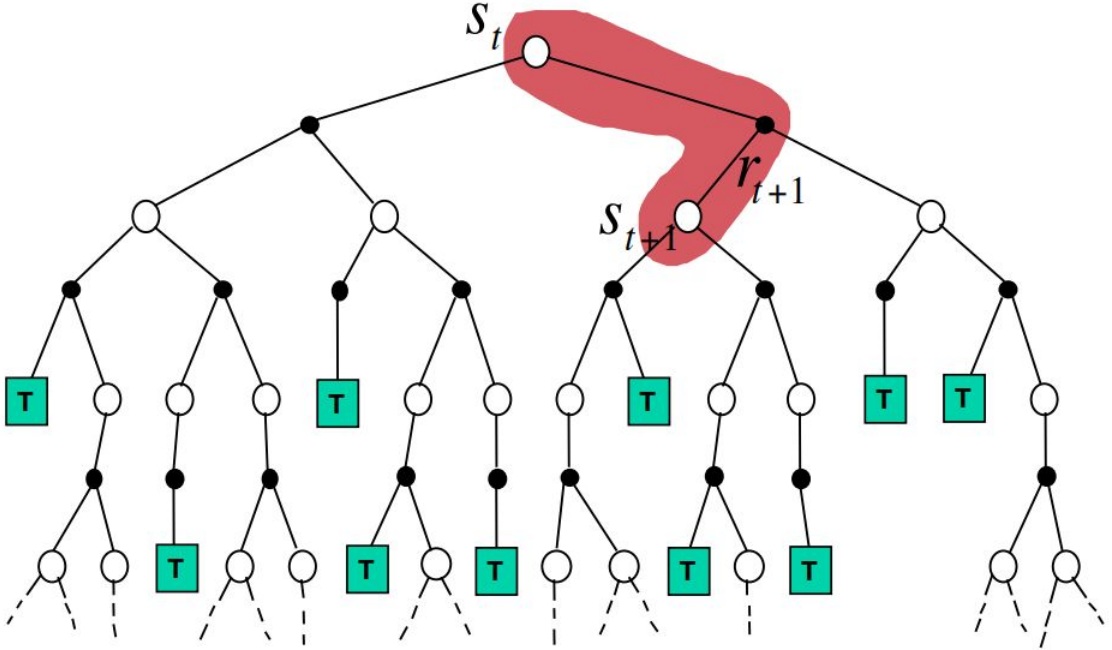




Additional infomation:
https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/

TD learning

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



Monte Carlo Method

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

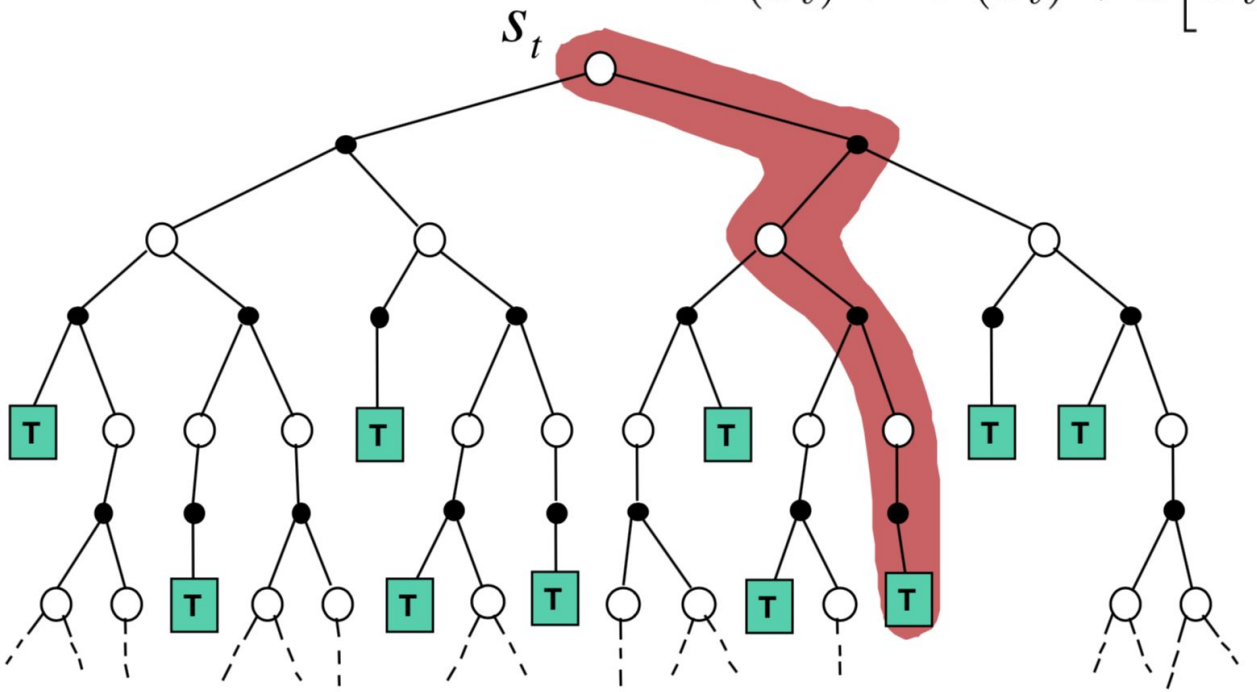
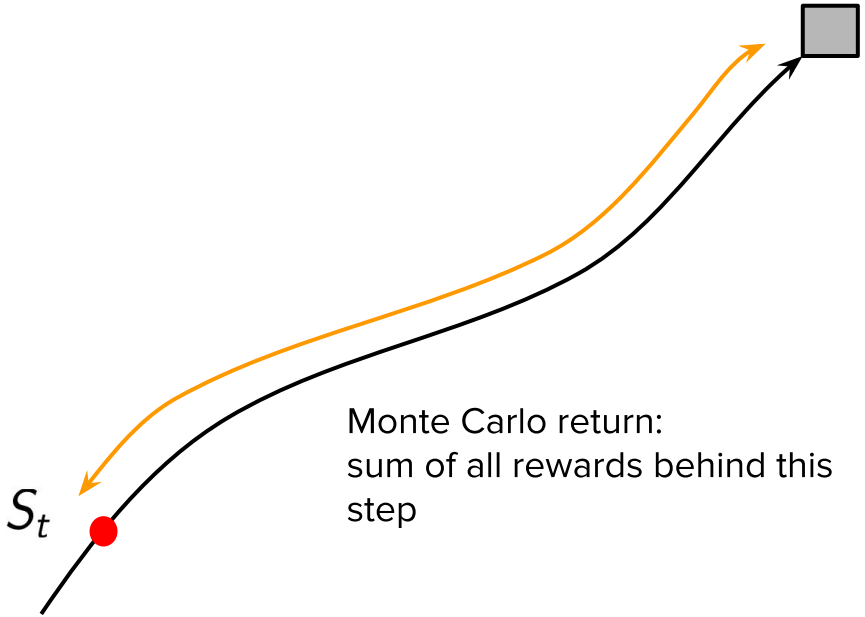


Figure source: David Silver's slides

Monte Carlo method

Monte Carlo update:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

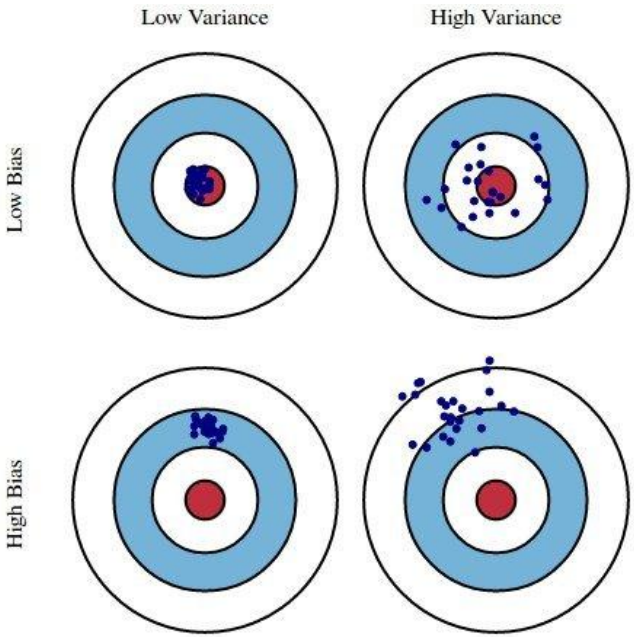


Monte Carlo method

Monte Carlo update:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

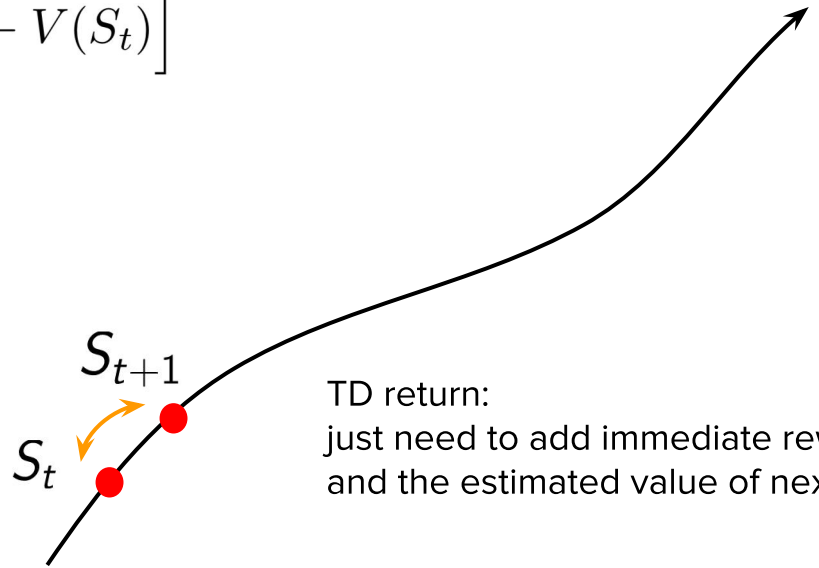
- Unbiased
  - The target is true return of the sample.
- High variance
  - The targets among samples are very different because they are depend on whole trajectory.



TD method

TD update:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$



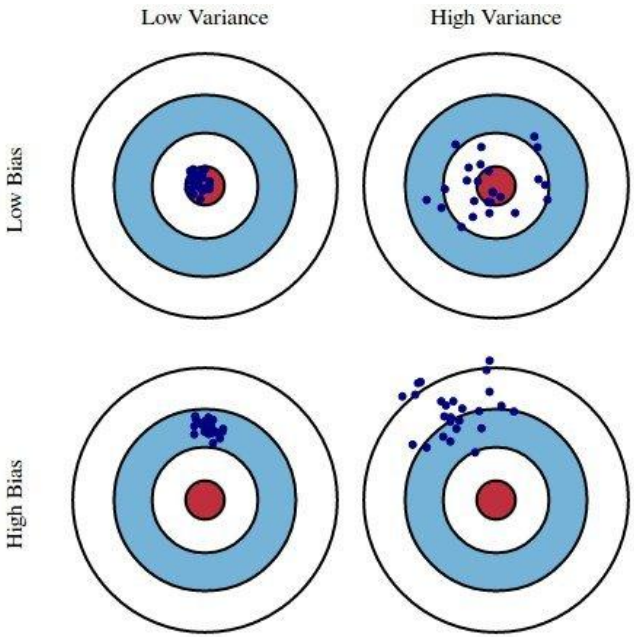
TD return:
just need to add immediate reward
and the estimated value of next state.

TD method

Monte Carlo update:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

- biased
  - The target is also an estimated value (because of V(s))
- low variance
  - The targets are slightly different, because it just take one-step.



Can we evaluate policy steps less than Monte Carlo but more than one-step TD?

n-step bootstrapping

- One-step TD learning:
  - target of TD:

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

  - bellman equation:

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

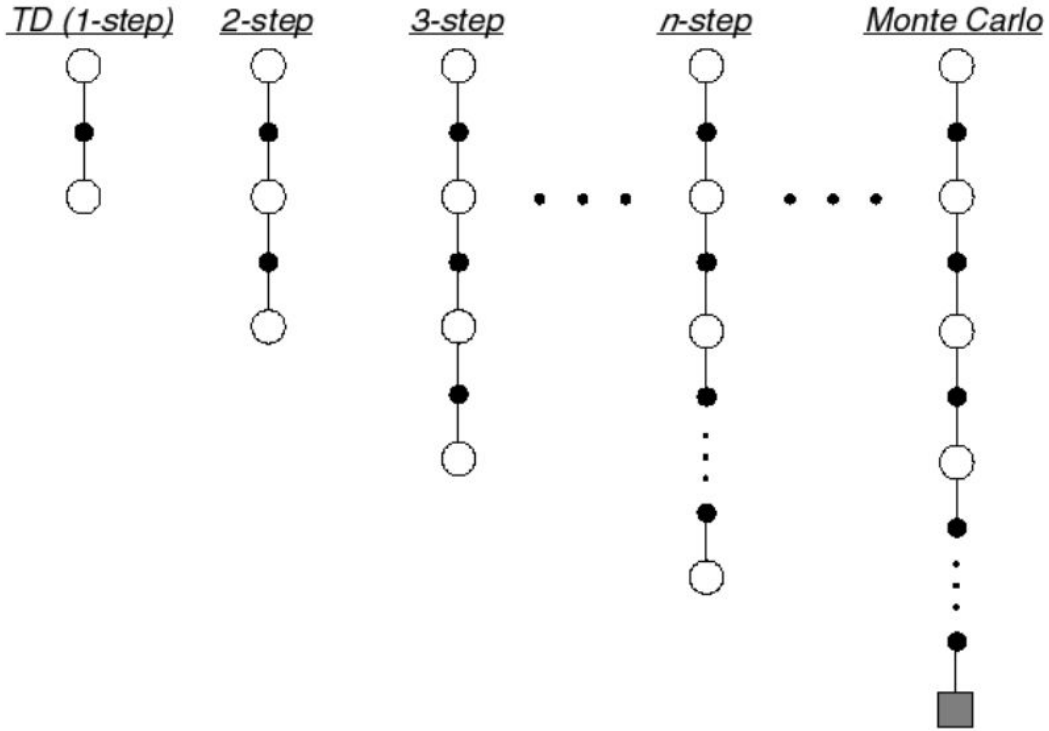
- Monte Carlo method:
  - target of MC:

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

  - bellman equation:

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t)\Big]$$

n-step TD



n-step TD

n-step TD return:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Bellman equation:

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

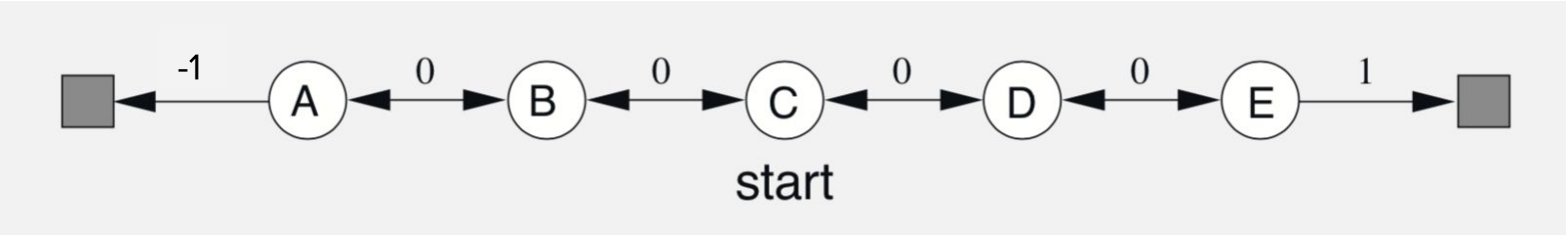
n-step TD for estimating $V \approx v_\pi$

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

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

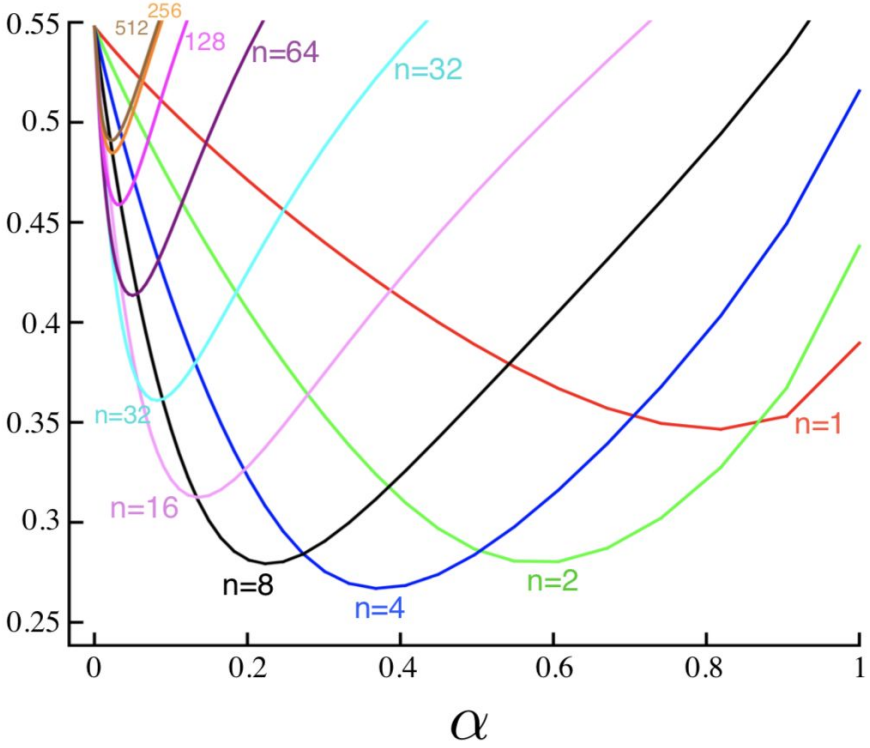
Performance of n-step TD: Random walk



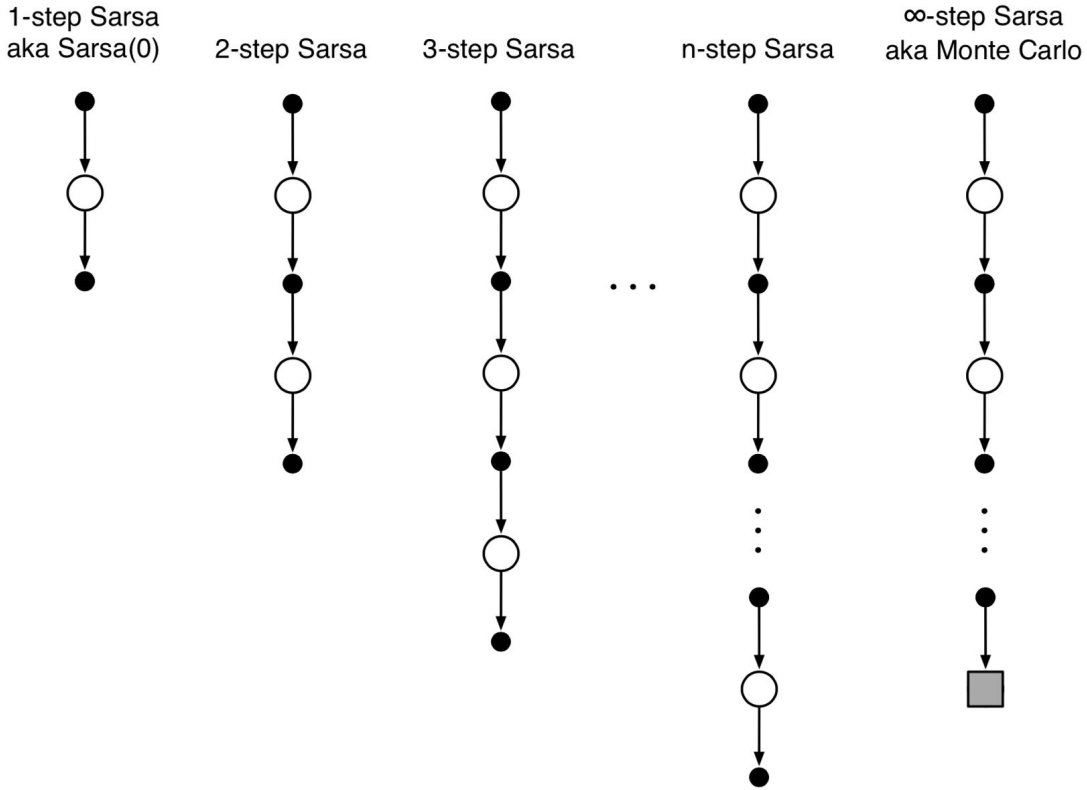
- 2 terminal states
- with 19 states instead of 5

Performance of n-step TD: Random walk

Average RMS error over 19 states and first 10 episodes



n-step Sarsa


1-step Sarsa
aka Sarsa(0)
2-step Sarsa
3-step Sarsa
n-step Sarsa
∞-step Sarsa
aka Monte Carlo

n-step Sarsa

n-step Sarsa return:

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

Bellman equation:

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

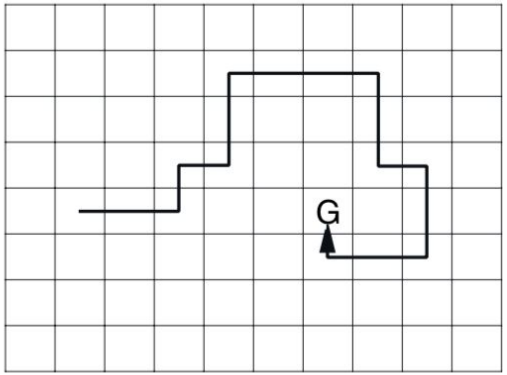
n-step Sarsa for estimating $Q \approx q_*$ or q_π

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 $n+1$

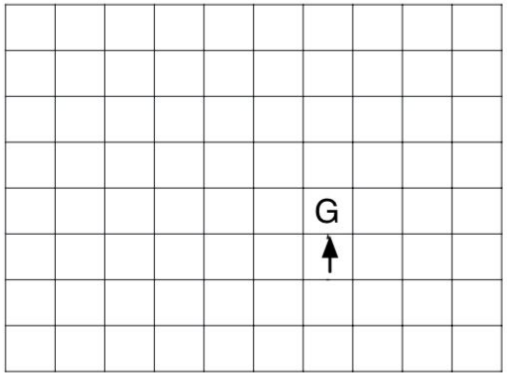
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T-1$

n-step Sarsa

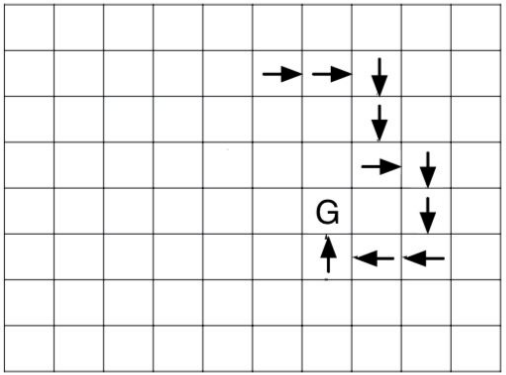
Path taken



Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa

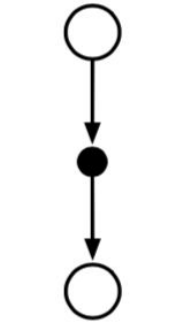


$TD(\lambda)$

In the previous slides, we have seen TD(0) before. What does the **0** mean?

one-step TD/TD(0) update:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

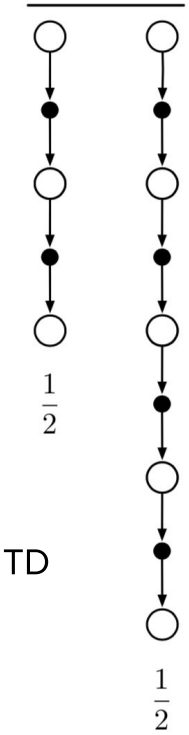


TD(0)

$TD(\lambda)$

We have already introduce n-step TD learning, like:
1-step, 3-step, 8-step etc. Maybe 4-step TD return is best for learning.

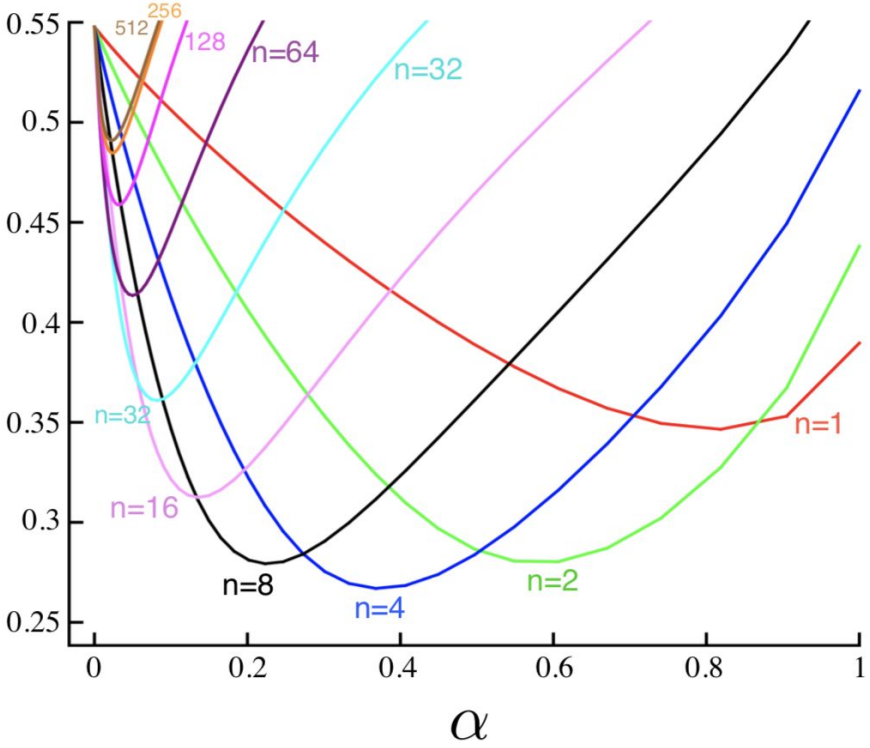
Can we combine them together to get better performance?



2-step TD & 4-step TD

Recap: Performance of n-step TD: Random walk

Average RMS error over 19 states and first 10 episodes



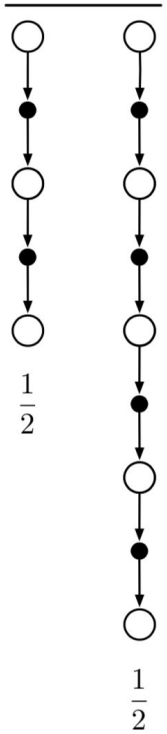
$TD(\lambda)$

A simple way to combine n-step TD returns is to average n-step return as long as the weights on component returns are positive and sum to 1.

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

$$G_{t:t+4} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 V(S_{t+4})$$

$$G_{avg} = \frac{1}{2} G_{t:t+2} + \frac{1}{2} G_{t:t+4}$$



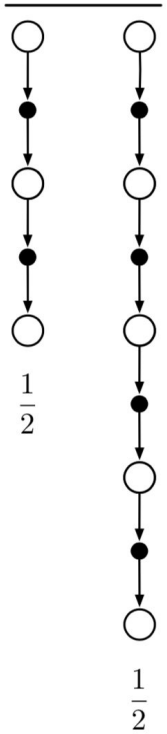
$TD(\lambda)$

A simple way to combine n-step TD returns is to average n-step return as long as the weights on component returns are positive and sum to 1.

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

$$G_{t:t+4} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 V(S_{t+4})$$

$$G_{avg} = \frac{1}{2} G_{t:t+2} + \frac{1}{2} G_{t:t+4}$$ → This is called compound return

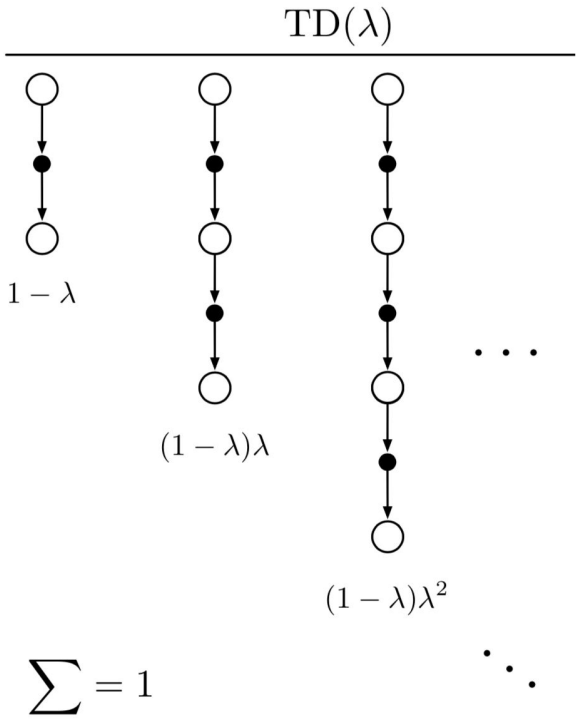


$TD(\lambda)$

The $TD(\lambda)$ is one particular way of averaging n-step updates.

This average contains **all** the n-step updates, each weighted proportional to λ^{n-1}

Besides, each term of n-step returns are normalized by a factor of $1-\lambda$ to ensure that the weights sum to 1.

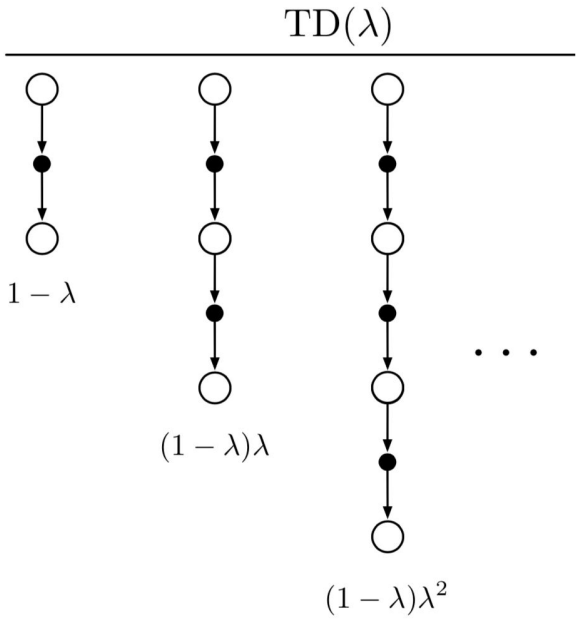


$TD(\lambda)$

The return of $TD(\lambda)$ is defined as following:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}$$

We called it $\lambda - return$



$$\sum = 1$$

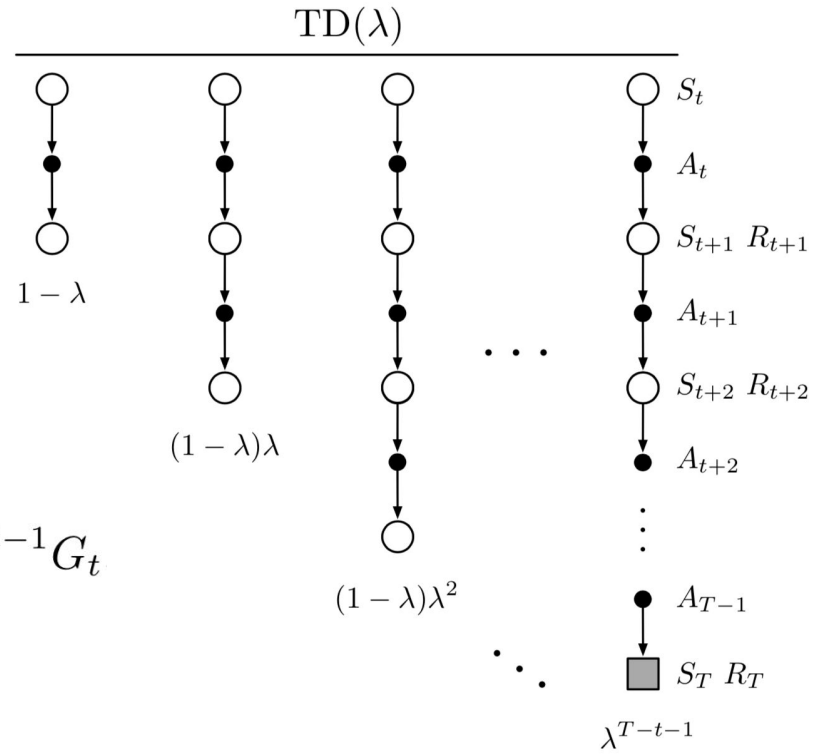
$$\frac{1-\lambda}{1-\lambda} = 1$$

$TD(\lambda)$

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}$$

Another form is to separate post-termination terms from the main sum.

$$G_t^\lambda \quad = \quad (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} \quad + \quad \lambda^{T-t-1}G_t$$



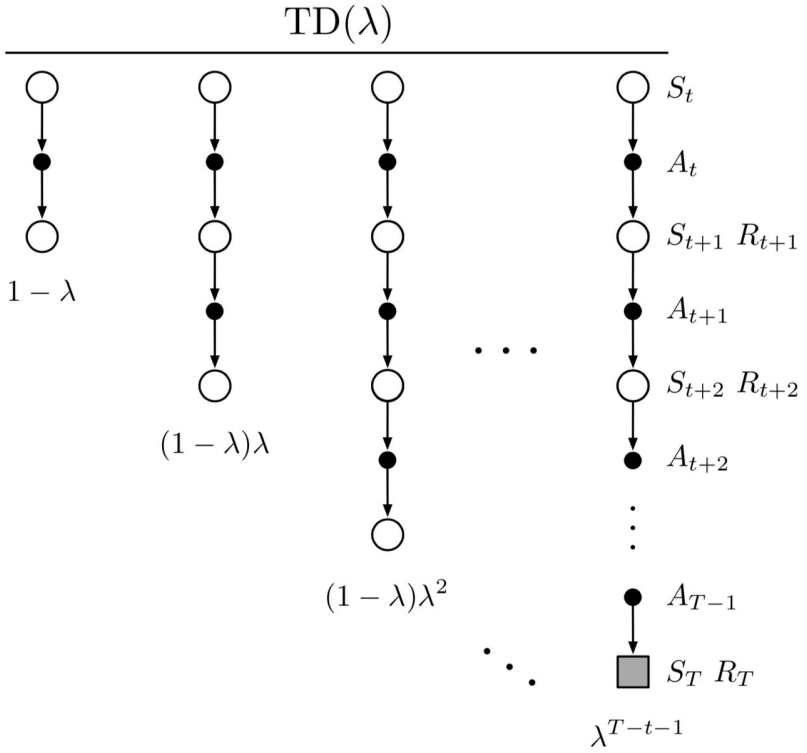
$TD(\lambda)$

$$G_t^\lambda \quad = \quad (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} \quad + \quad \lambda^{T-t-1} G_t$$

- Where

$\lambda = 0$ *is* $TD(0)$ / one-step TD

$\lambda = 1$ *is* $TD(1)$ / Monte Carlo



n-step backups

- Backup (on-line or off-line):

$$\Delta V_t(s_t) = \alpha\left[R_t^{(n)} - V_t(s_t)\right]$$

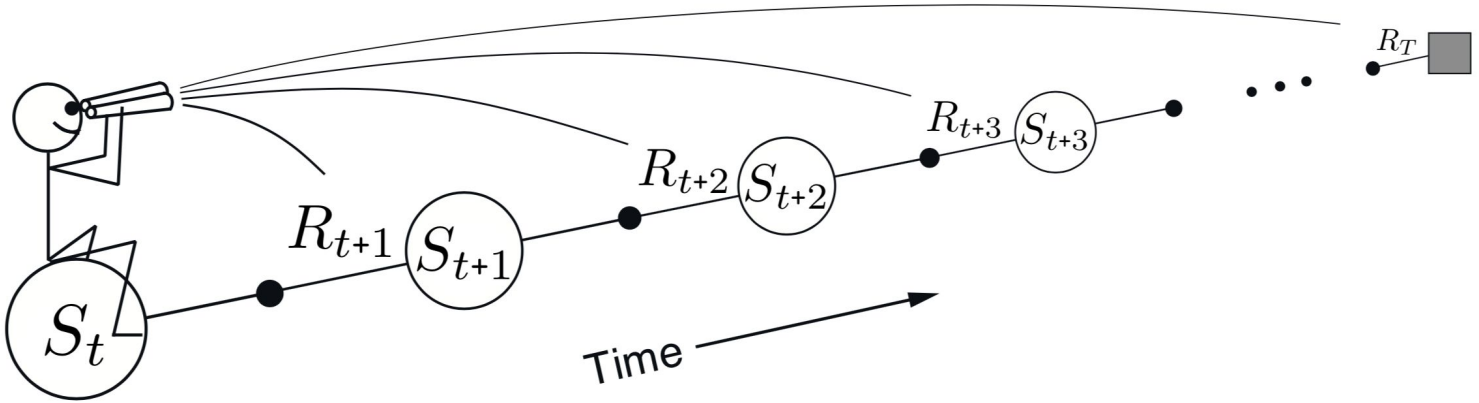
- Off-line: the increments are accumulated "on the side" and are not used to change value estimates until the end of the episode.

$$V(s) := V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$$

- On-line: the updates are done during the episode, as soon as the increment is computed.

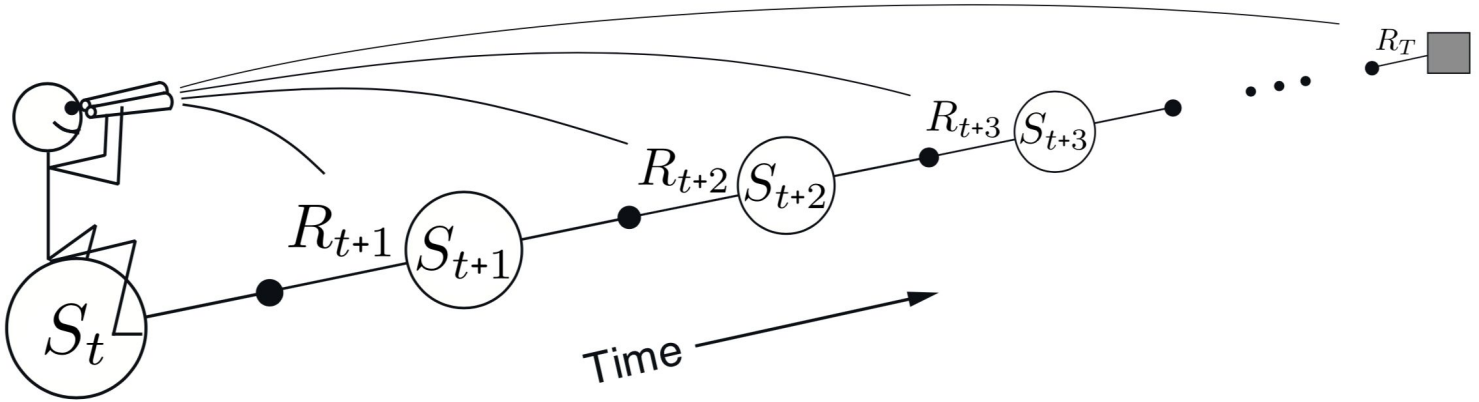
$$V_{t+1}(s) := V_t(s) + \Delta V_t(s)$$

Forward $TD(\lambda)$



- Update value function towards the $\lambda - return$
- Foward view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes (Off-line learning)

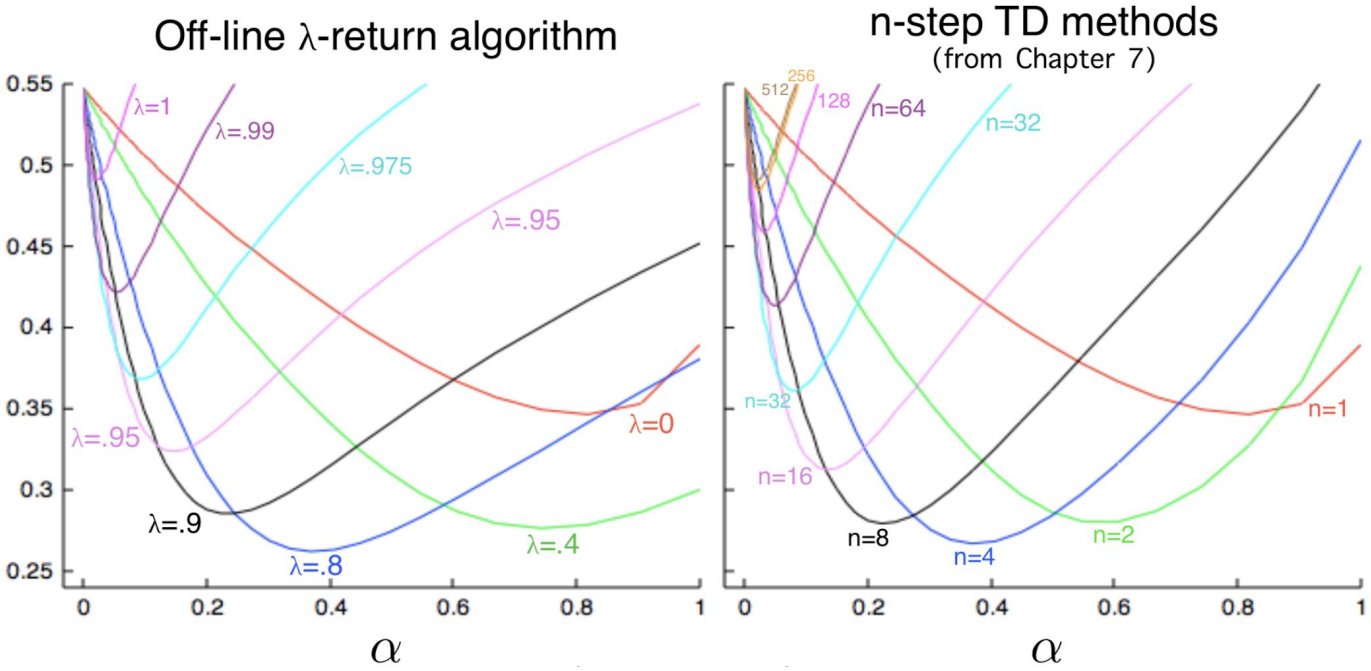
Forward $TD(\lambda)$



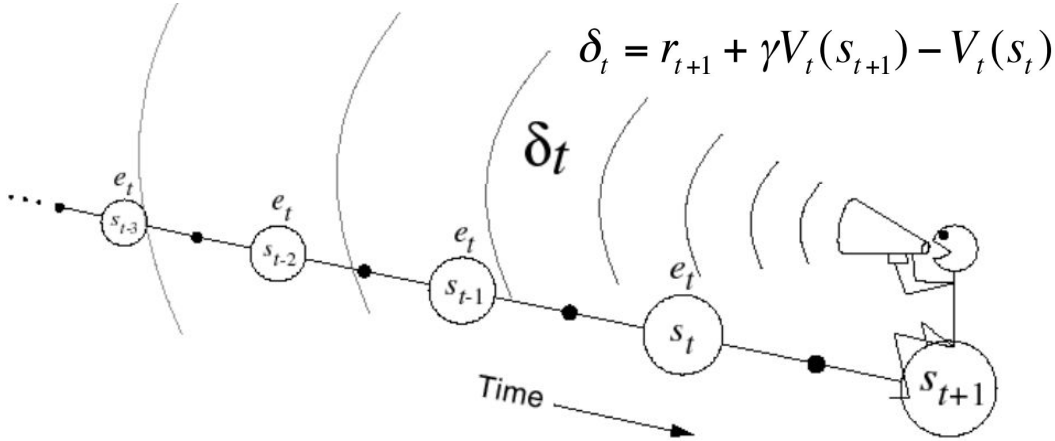
In forward view, after looking forward from and updating one state, we move on to the next and **never** have to work with the preceding state agein.

$TD(\lambda)$ vs N-step TD: 19-state random walk

RMS error at the end of the episode over the first 10 episodes



Backward $TD(\lambda)$



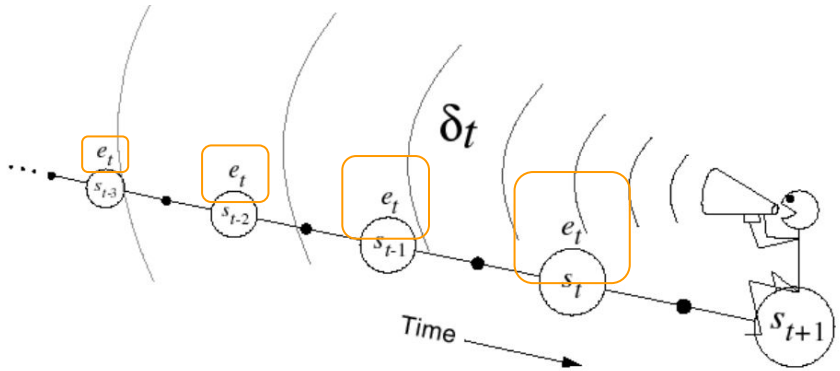
- Forward view provides theory
- Backward view provides mechanism
- Shout δ_t backward over time
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

Backward $TD(\lambda)$

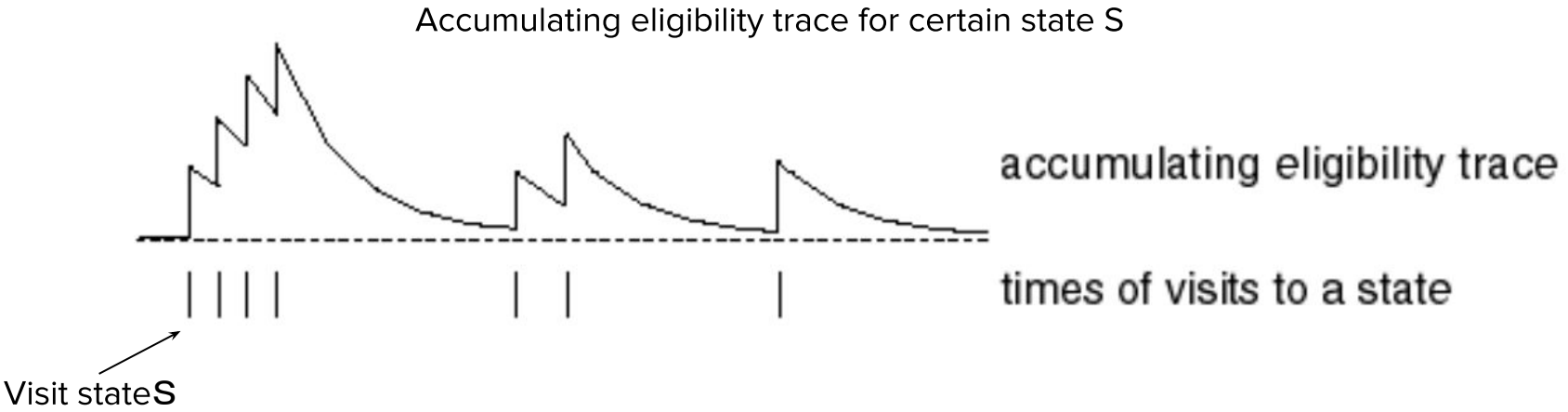
Eligibility trace denoted by $E_0(s)$, which keeps track the weights of updating value function for every state.

$$E_0(s) = 0$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$



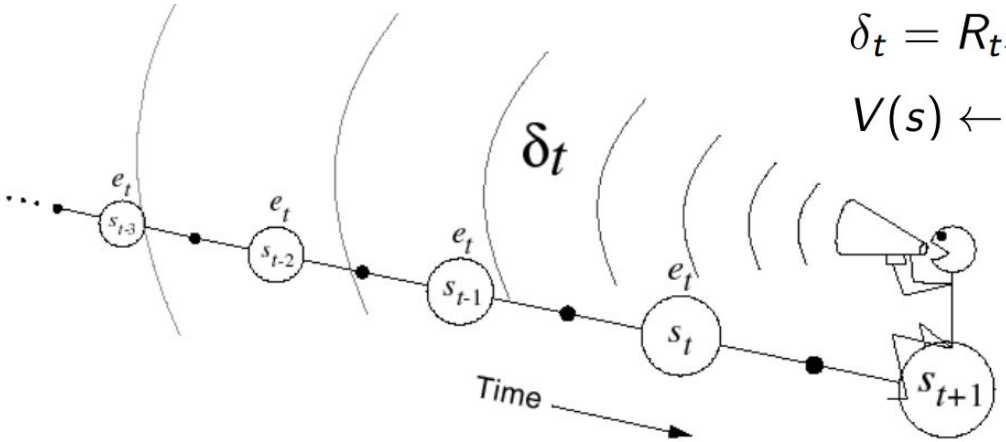
Eligibility Trace



From David's slides

Backward $TD(\lambda)$

- Keep an eligibility trace for every state s
- Update value V(s) for every state s in the single step
- In proportion to TD-error δ_t and eligibility trace $E_t(s)$



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

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

From David's slides

$TD(0)$ and $TD(\lambda)$

- When $\lambda = 0$, only the current state is updated

$$E_t(s) = \mathbf{1}(S_t = s)$$
$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

$$E_0(s) = 0$$
$$E_t(s) = \gamma \lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$

- This exactly equivalent to TD(0) update

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

Telescoping in TD(1)

When $\lambda = 1$, sum of TD errors telescopes into MC error,

$$
\begin{aligned}
& \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \ldots + \gamma^{T-1-t}\delta_{T-1} \\
&= R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \\
&+ \gamma R_{t+2} + \gamma^2 V(S_{t+2}) - \gamma V(S_{t+1}) \\
&+ \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3}) - \gamma^2 V(S_{t+2}) \\
&\quad \vdots \\
&+ \gamma^{T-1-t} R_T + \gamma^{T-t} V(S_T) - \gamma^{T-1-t} V(S_{T-1}) \\
&= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots + \gamma^{T-1-t} R_T - V(S_t) \\
&= G_t - V(S_t)
\end{aligned}
$$

Online Tabular $TD(\lambda)$

Initialize $V(s)$ arbitrarily

Repeat (for each episode):

$e(s) = 0$, for all $s \in S$

Initialize s

Repeat (for each step of episode):

$a \leftarrow$ action given by π for s

Take action a, observe reward, r, and next state s'

$\delta \leftarrow r + \gamma V(s') - V(s)$

$e(s) \leftarrow e(s) + 1$

For all s:

$V(s) \leftarrow V(s) + \alpha\delta e(s)$

$e(s) \leftarrow \gamma\lambda e(s)$

$s \leftarrow s'$

Until s is terminal

Online Tabular $TD(\lambda)$

Initialize $V(s)$ arbitrarily

Repeat (for each episode):

$e(s) = 0$, for all $s \in S$

Initialize s

Repeat (for each step of episode):

$a \leftarrow$ action given by π for s

Take action a, observe reward, r, and next state s'

$\delta \leftarrow r + \gamma V(s') - V(s)$

$e(s) \leftarrow e(s) + 1$

For all s:

$V(s) \leftarrow V(s) + \alpha\delta e(s)$

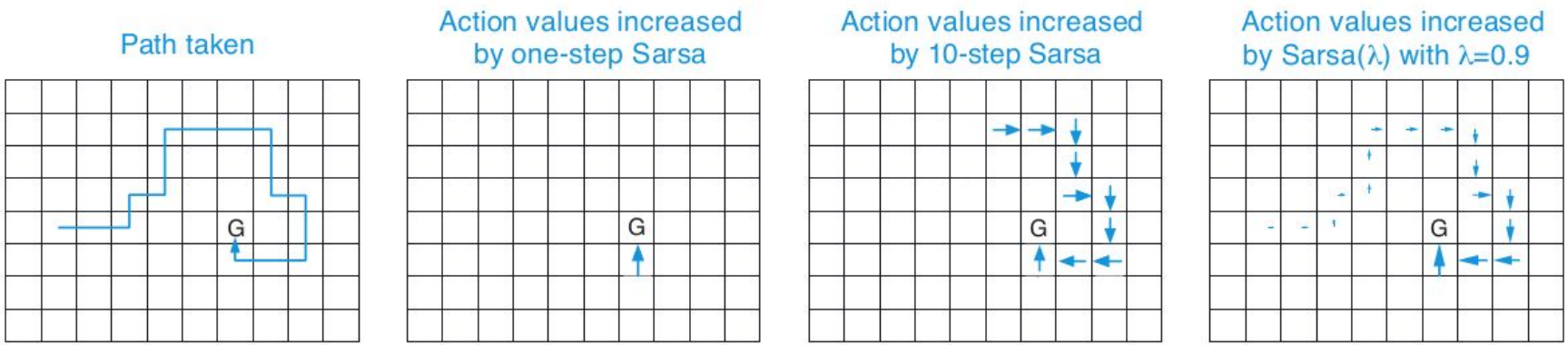
$e(s) \leftarrow \gamma\lambda e(s)$

$s \leftarrow s'$

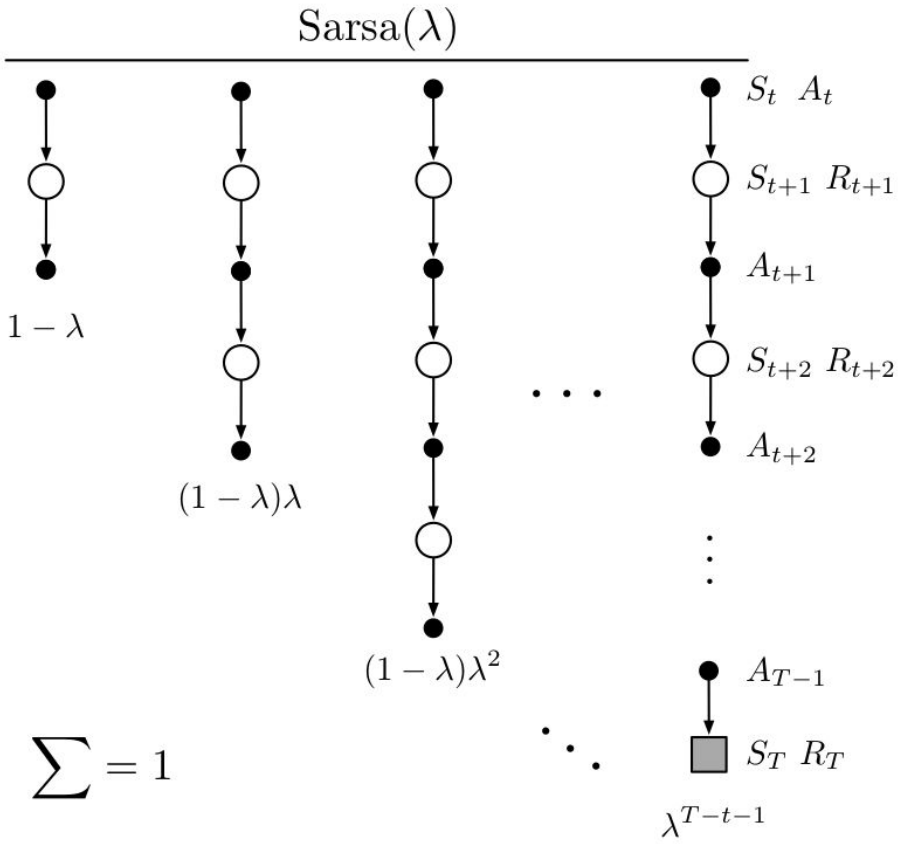
Until s is terminal

Backward part !

Sarsa(λ)



Sarsa(λ)



Sarsa(λ)

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
      $E(s,a) \leftarrow \gamma \lambda E(s,a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

From David's slides

Off-line update VS On-line update

Off-line updates

- updates are accumulated within episode
- but applied in batch at the end of episode

On-line updates

- updates are applied online at each step within episode
- can be applied in continuing task

Cons of Tabular method

In the previous method, we use a large table to store the value of each state or state-action pair which is called **tabular method**.

In real scene, there are too many state-action pairs to store. Besides, the state /observation would also be much complicated, for example: an image with high resolution. It causes **curse of dimensionality**.

Cons of Tabular method

In the previous method, we use a large table to store the value of each state or state-action pair which is called **tabular method**.

In real scene, there are too many state-action pairs to store. Besides, the state /observation would also be much complicated, for example: an image with high resolution. It causes **curse of dimensionality**.

> We can use function approximator to estimate the value function, and it has generalizability!

Relationship between DP and TD

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

From David's slides

On-policy & Off-policy 補充

On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data.

Recommended Papers

1. John et al., High-Dimensional Continuous Control Using Generalized Advantage Estimation (ICLR 2016)
2. Kristopher et al., Multi-step Reinforcement Learning: A Unifying Algorithm (AAAI 2018)

Reference

1. Sutton’s textbook Chapter 6, 7, 12
2. Reinforcement Learning Lecture4 from UCL
3. Künstliche Intelligenz’s slides: https://www.tu-chemnitz.de/informatik/KI/scripts/ws0910/ml09_7.pdf