Reinforcement Learning

Computer Engineering Department
Sharif University of Technology

Mohammad Hossein Rohban, Ph.D.

Hossein Hasani

Spring 2023

Courtesy: Some slides are adopted from CS 285 Berkeley, and CS 234 Stanford, and Pieter Abbeel's compact series on RL.

Outline

- Recap of Monte Carlo
- Temporal Difference Learning (Prediction)
- TD vs. MC
- n-Step TD
- TD(λ) (Forward and Backward view)
- Temporal Difference Learning (Control)
- SARSA
- On and Off-Policy Learning
- Q-Learning

Disadvantages of MC Learning

- We have seen MC algorithms can be used to learn value predictions
- But when episodes are long, learning can be slow
  - ...we have to **wait until an episode ends** before we can learn
  - ...return can have **high variance**
- Are there alternatives? (Yes)

Temporal Difference Learning

Prediction

TD Overview

TD methods learn directly from episodes of experience

TD is *model-free*: no knowledge of MDP transitions / rewards

TD learns from *incomplete* episodes, by *bootstrapping*

TD updates a guess towards a guess

Temporal Difference Learning by Sampling Bellman Equations

- Bellman update equations:

$$v_{k+1}(s) = \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\right]$$

- We can sample this!

$$v_{t+1}(S_t) = R_{t+1} + \gamma v_t(S_{t+1})$$

- Samples could be averaged, in a similar way to MC:

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t \underbrace{\left(\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t) \right)}_{\text{temporal difference error } \delta_t}$$

Temporal Difference Learning

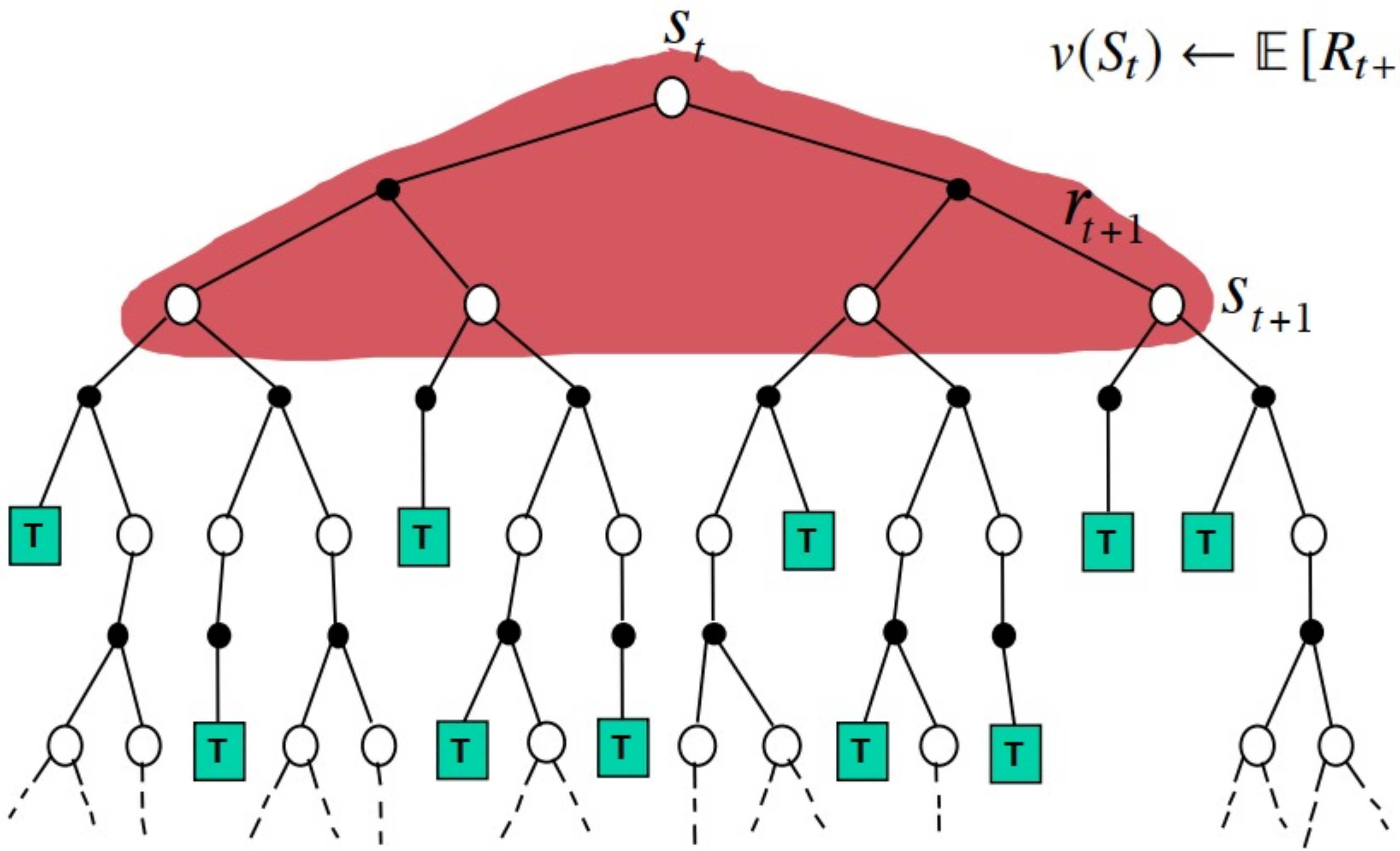
- Prediction setting: learn v_π online from experience under policy π
- Monte Carlo
  - Update value $v_n(S_t)$ towards sampled return G_t

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\,(G_t - v_n(S_t))$$

- TD Learning
  - Update value $v_t(S_t)$ towards estimated return $R_{t+1} + \gamma v(S_{t+1})$

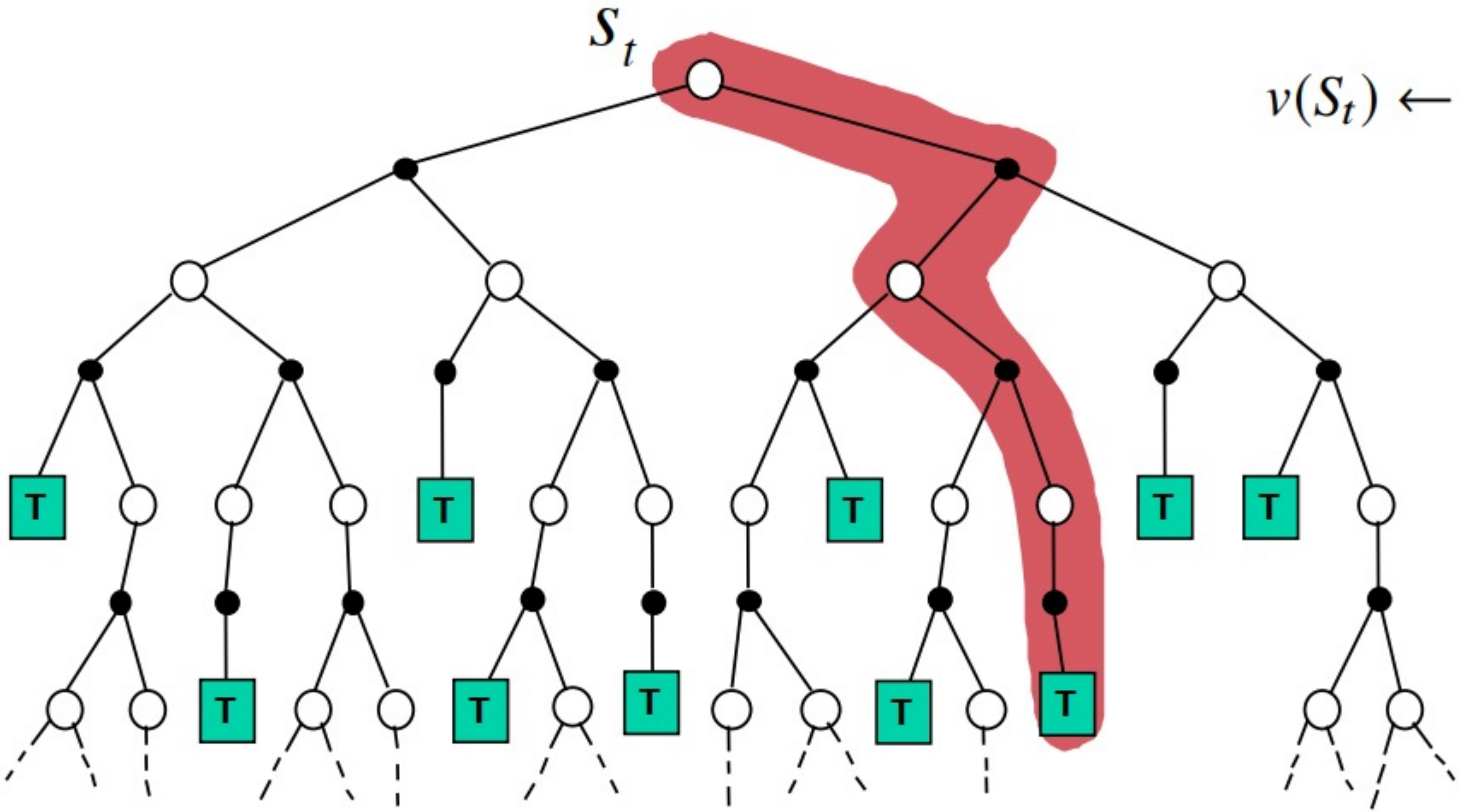
$$v_{t+1}(S_t) \leftarrow v_t(S_t) + \alpha \left(\overbrace{\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t)}^{\text{TD error}} \right)$$

Backup (Dynamic Programming)



$$v(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)\right]$$

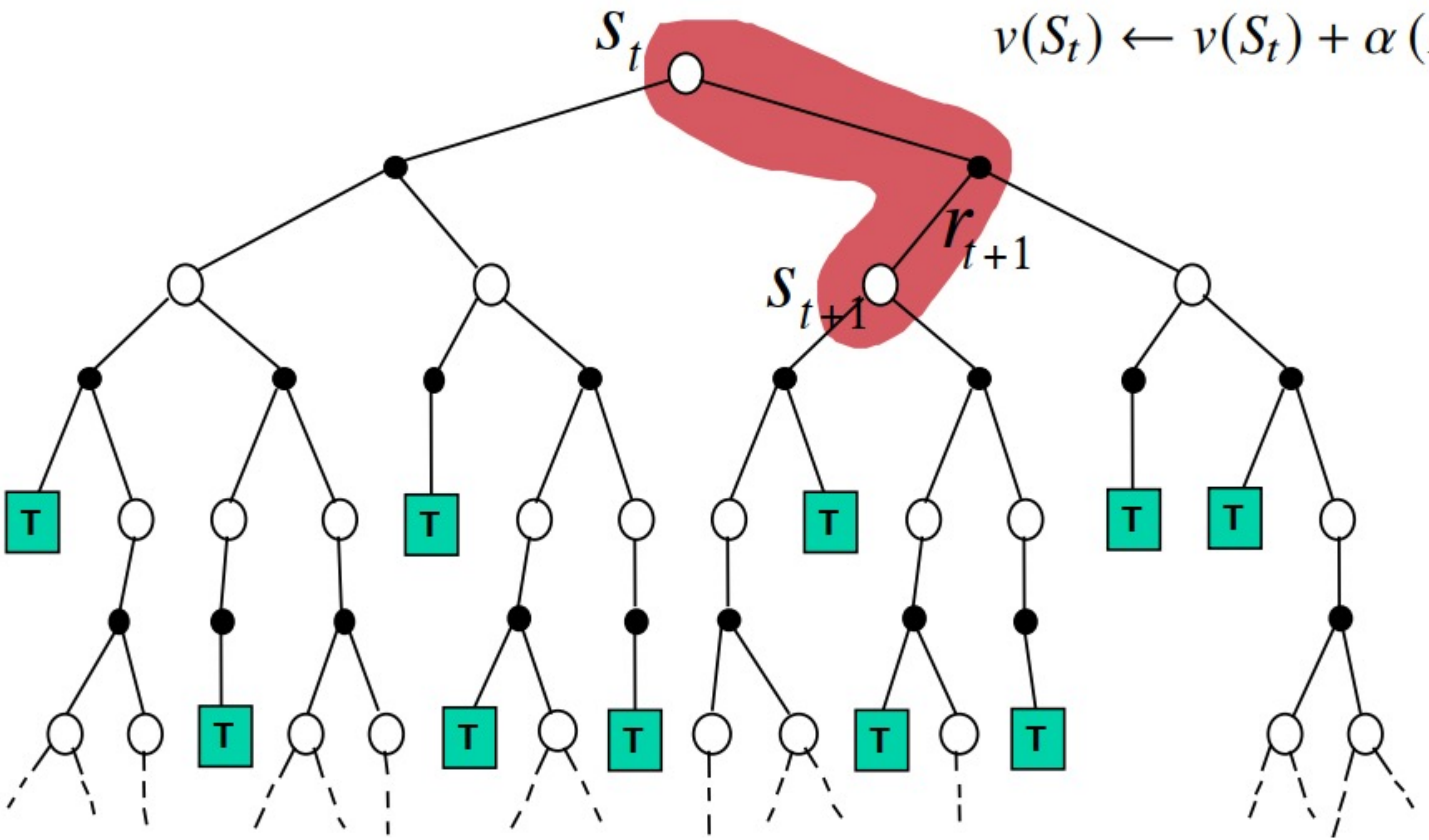
Backup (Monte Carlo)



$$v(S_t) \leftarrow v(S_t) + \alpha\,(G_t - v(S_t))$$

Backup (Temporal Difference)

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



Bootstrapping and Sampling

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

TD Learning for action values

- We can apply the same idea to action values
- Temporal-difference learning for action values:
  - Update value $q_t(S_t, A_t)$ towards estimated return $R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$

$$q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha \left(\overbrace{\underbrace{R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1})}_{\text{target}} - q_t(S_t, A_t)}^{\text{TD error}} \right)$$

TD vs. MC

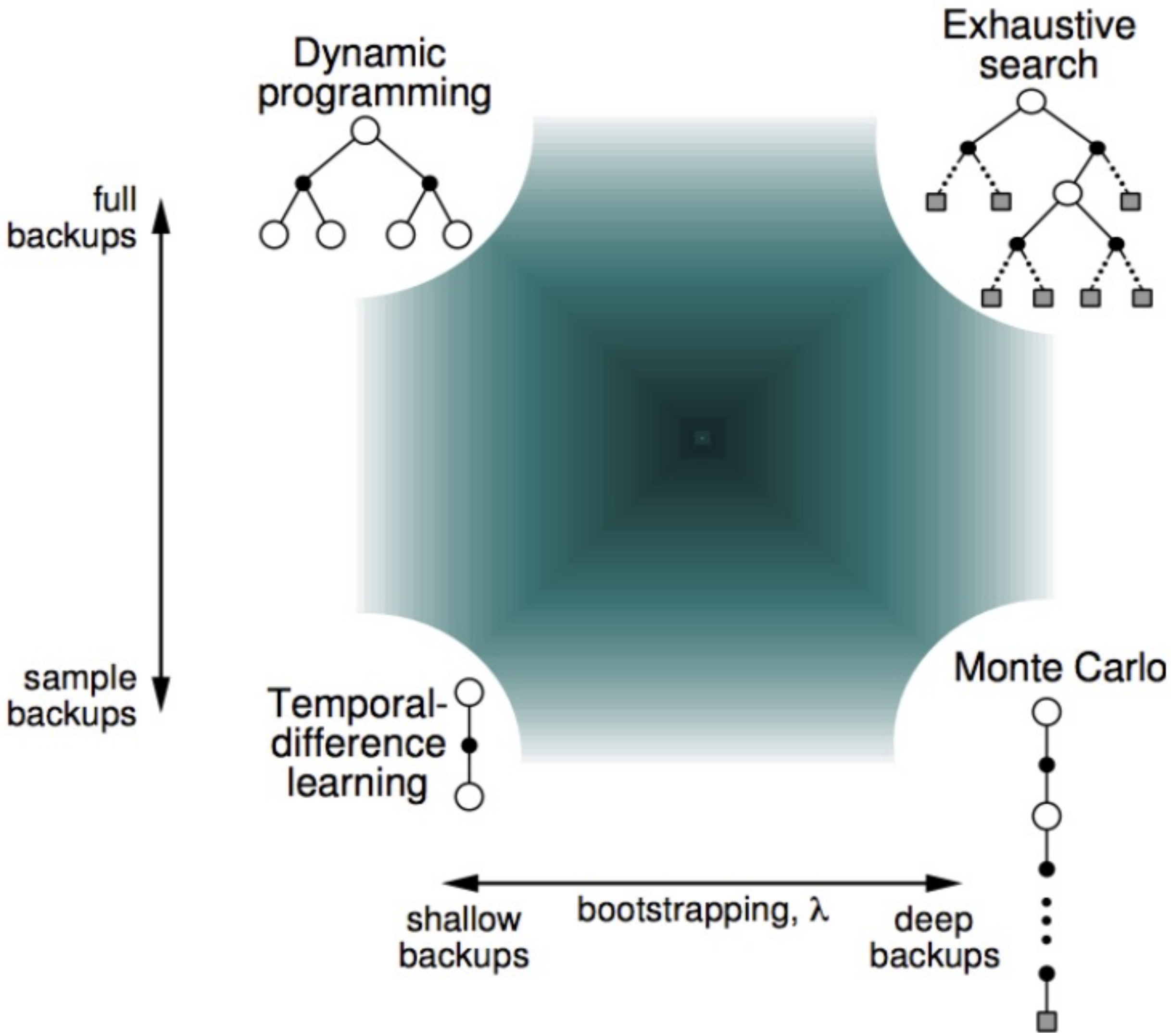
- **TD can learn before knowing the final outcome**
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- **TD can learn without the final outcome**
  - MC must wait until end of episode before return is known
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments
- **TD is independent of the temporal span of the prediction**
  - TD can learn from single transitions
  - MC must store all predictions (or states) to update at the end of an episode
- **TD needs reasonable value estimates**

Bias/Variance Tradeoff

- MC return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots$ is an unbiased estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma v_t(S_{t+1})$ is a biased estimate of $v_\pi(S_t)$
  - unless $\mathbb{E}[v_t(S_{t+1})|S_{t+1}] = v_\pi(S_{t+1})$
- But the TD target has lower variance:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward

Between MC and TD: Multi-step TD

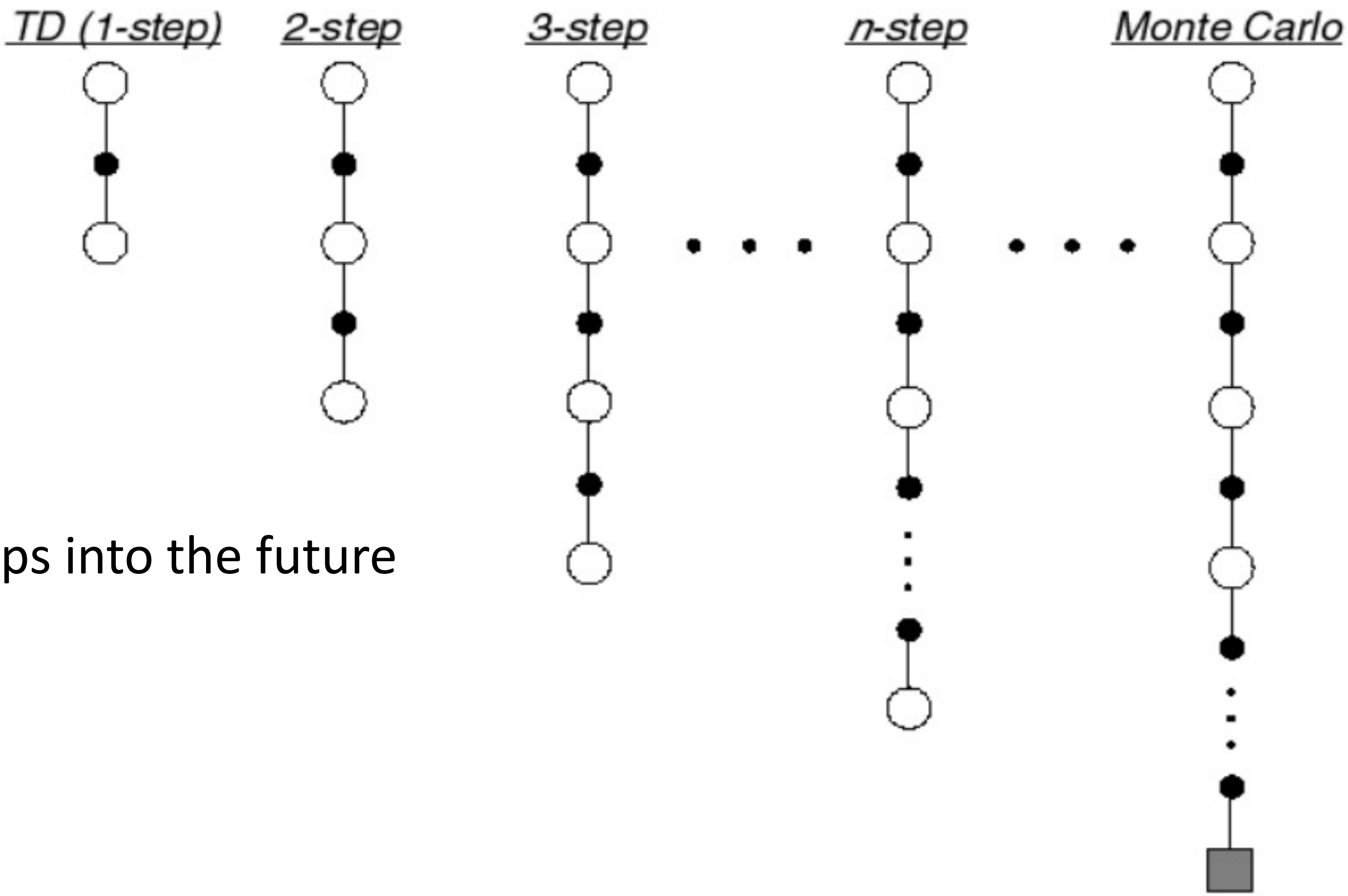
General View of Reinforcement Learning



Motivation

- TD uses value estimates which might be inaccurate
- In addition, information can propagate back quite slowly
- In MC information propagates faster, but the updates are noisier
- We can go in between TD and MC

n-Step Prediction



- Let TD target look n steps into the future

n-Step Returns

- Consider the following n-step returns for n = 1; 2; …:

$$
\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T
\end{array}
$$

- In general, the n-step return is defined by

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

- Multi-step temporal-difference learning

$$v(S_t) \leftarrow v(S_t) + \alpha \left(G_t^{(n)} - v(S_t) \right)$$

λ-Returns

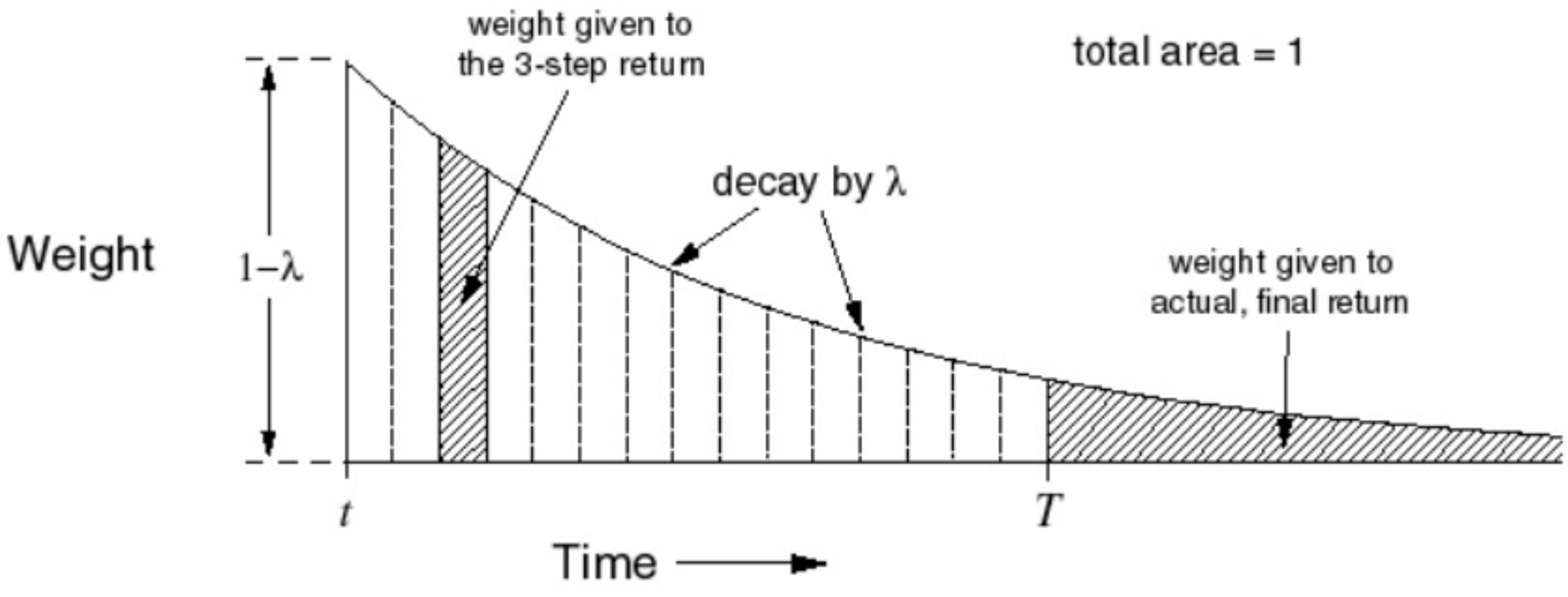
- The λ-return G_t^{λ} combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Forward view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right)$$

TD(λ) Weighting Function



$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

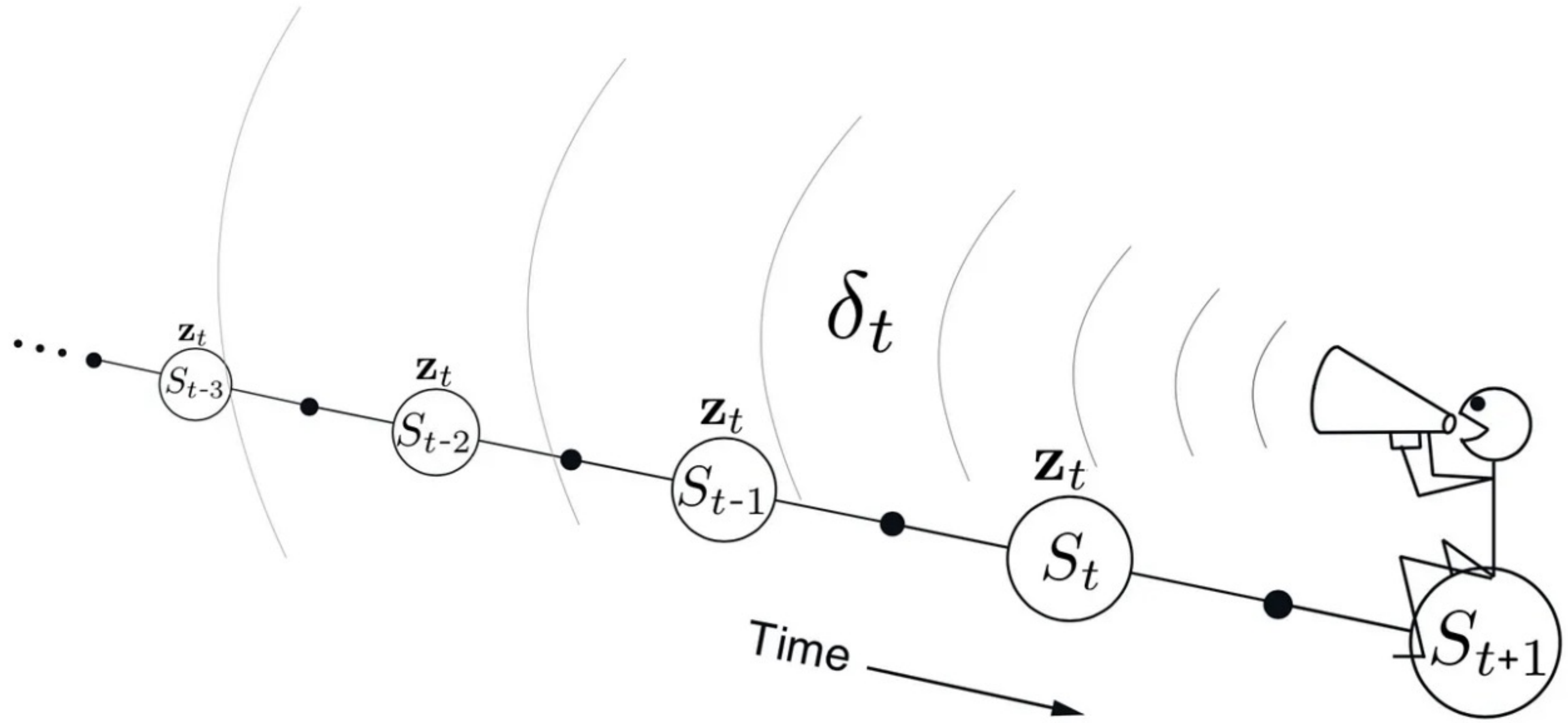
Forward-view TD(λ)

- Update value function towards the λ-return
- Forward-view looks into the future to compute G_t^{λ}
- Like MC, can only be computed from complete episodes

Backward-view TD(λ)

- Forward view provides theory
- Backward view provides mechanism
- Update online, every step, from incomplete sequences

Backward-view TD(λ)



Backward-view TD(λ)

- Keep an eligibility trace for every state s

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

- Update value V (s) for every state s in proportion to TD-error δ_t and eligibility trace $E_t(s)$

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

TD(λ) and TD(0)

- When λ = 0, only current state is updated

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

- This is exactly equivalent to TD(0) update

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

Online and Offline Updates

- Offline updates
  - Updates are accumulated within episode but applied in batch at the end of episode
- Online updates
  - updates are applied online at each step within episode

Theorem

The sum of offline updates is identical for forward-view and backward-view TD(λ)

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t^{\lambda} - V(S_t) \right) \mathbf{1}(S_t = s)$$

Equivalence of Forward and Backward TD in Online and Offline

Offline updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\Vert$	$\Vert$	$\Vert$
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\Vert$	$\nparallel$	$\nparallel$
Forward view	TD(0)	Forward TD(λ)	MC
	$\Vert$	$\Vert$	$\Vert$
Exact Online	TD(0)	Exact Online TD(λ)	Exact Online TD(1)

$=$ here indicates equivalence in total update at end of episode.

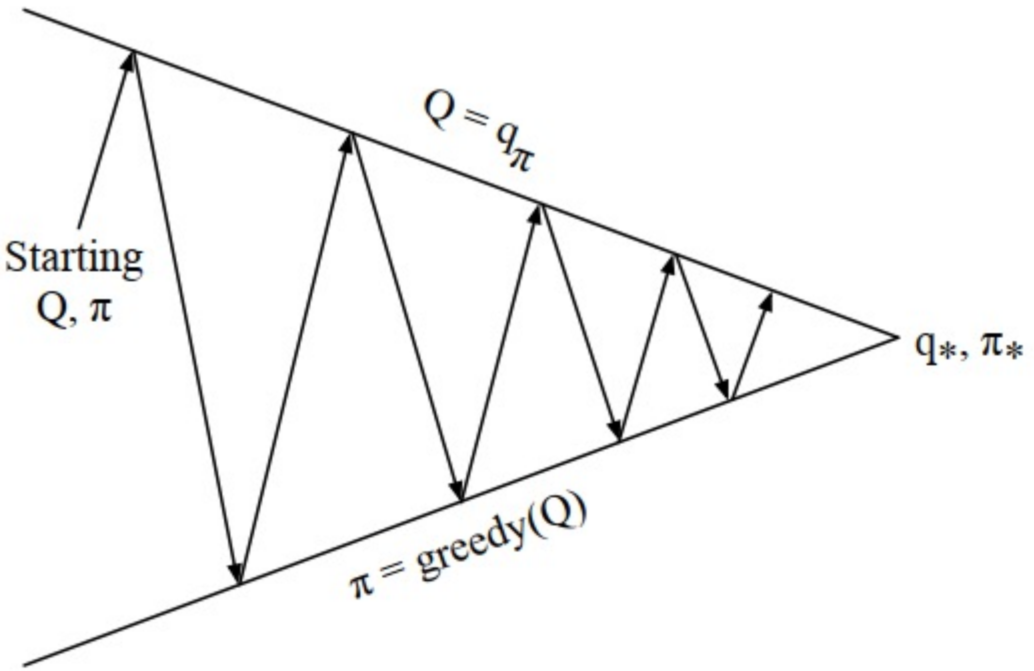
Temporal Difference Learning

Control

On and Off-Policy Learning

- On-policy learning
  - "Learn on the job“
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone’s shoulder“
  - Learn about policy π from experience sampled from μ

Generalized Policy Iteration with MC Evaluation (Review)



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

ε-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability 1 − ε choose the greedy action
- With probability ε choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

ε-Greedy Policy Improvement

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

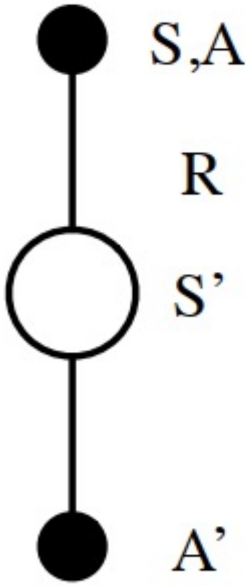
$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\
&= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\
&\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s)
\end{aligned}
$$

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

MC vs. TD Control

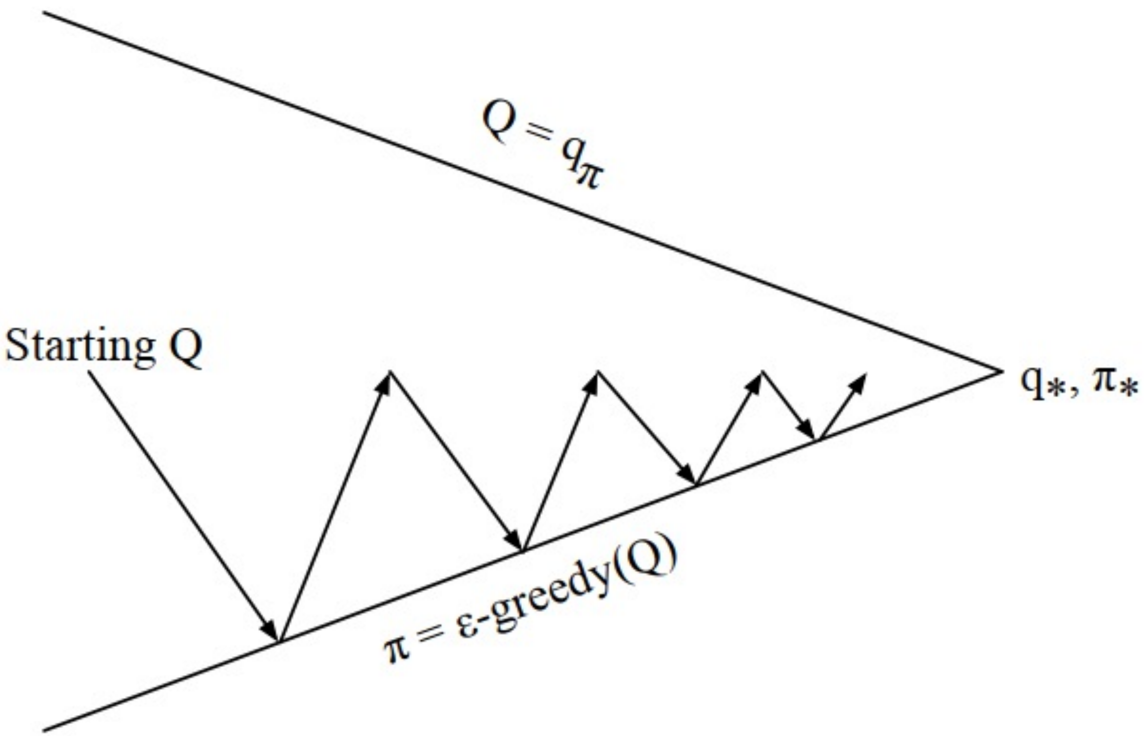
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Natural idea: use TD instead of MC in our control loop
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to Q(S, A)
  - Use ε-greedy policy improvement
  - Update every time-step

Updating Action-Value with SARSA



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

On-Policy Control with SARSA



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

SARSA Algorithm for On-Policy Control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

n-Step SARSA

Consider the following n-step returns for $n = 1, 2, \infty$:

$$
\begin{array}{lll}
n = 1 & (Sarsa) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\
n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{array}
$$

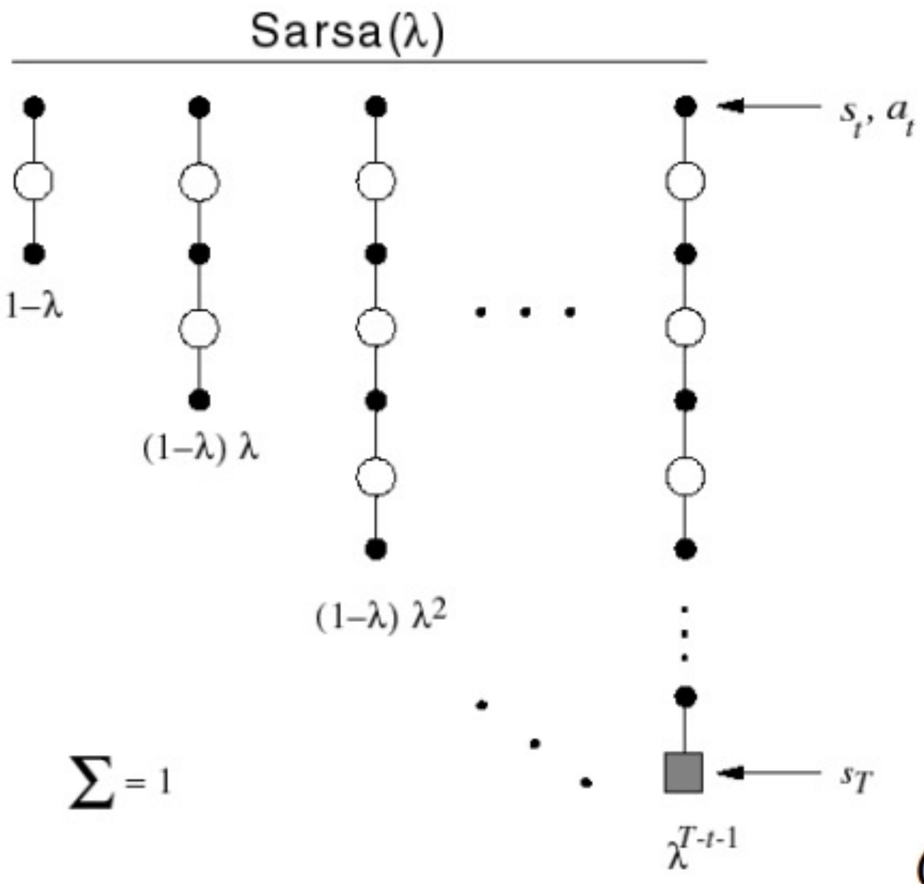
Define the n-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step Sarsa updates $Q(s, a)$ towards the n-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$$

Forward View SARSA(λ)



The q^λ *return* combines all n-step Q-returns $q_t^{(n)}$

Using weight $(1-\lambda)\lambda^{n-1}$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

Forward-view Sarsa(λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(q_t^\lambda - Q(S_t, A_t)\right)$$

Backward View SARSA(λ)

- Just like TD(λ), we use eligibility traces in an online algorithm
- But SARSA(λ) has one eligibility trace for each state-action pair

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

- Q(s, a) is updated for every state s and action a in proportion to TD-error δ_t and eligibility trace $E_t(s,a)$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

SARSA(λ) Algorithm

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

Off-Policy TD and Q-Learning

On and Off-Policy Learning

- On-policy learning
  - Learn about behavior policy π from experience sampled from π.
- Off-policy learning
  - Learn about target policy π from experience sampled from μ.
  - Learn ‘counterfactually’ about other things you could do: “what if...?”
  - e.g., “What if I would turn left?” =) new observations, rewards?

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$

- While using behavior policy $\mu(a,s)$ to generate actions

- Why is this important?

    - Learn from observing humans or other agents (e.g., from logged data)
    - Re-use experience from old policies (e.g., from your own past experience)
    - Learn about multiple policies while following one policy
    - Learn about greedy policy while following exploratory policy

Q-Learning

- Q-learning estimates the value of the greedy policy

$$q_{t+1}(s, a) = q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t) \right)$$

- Acting greedy all the time would not explore sufficiently

Theorem

Q-learning control converges to the optimal action-value function, $q \to q^*$, as long as we take each action in each state infinitely often.

- Note: no need for greedy behavior!
- Works for **any** policy that eventually selects all actions sufficiently often

Q-Learning for Off-Policy Control

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$;
  until S is terminal