

Reinforcement Learning

Computer Engineering Department
Sharif University of Technology

Mohammad Hossein Rohban, Ph.D.

Spring 2024

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
- $\text{TD}(\lambda)$ (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:

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

- 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:

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

temporal difference error δ_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

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

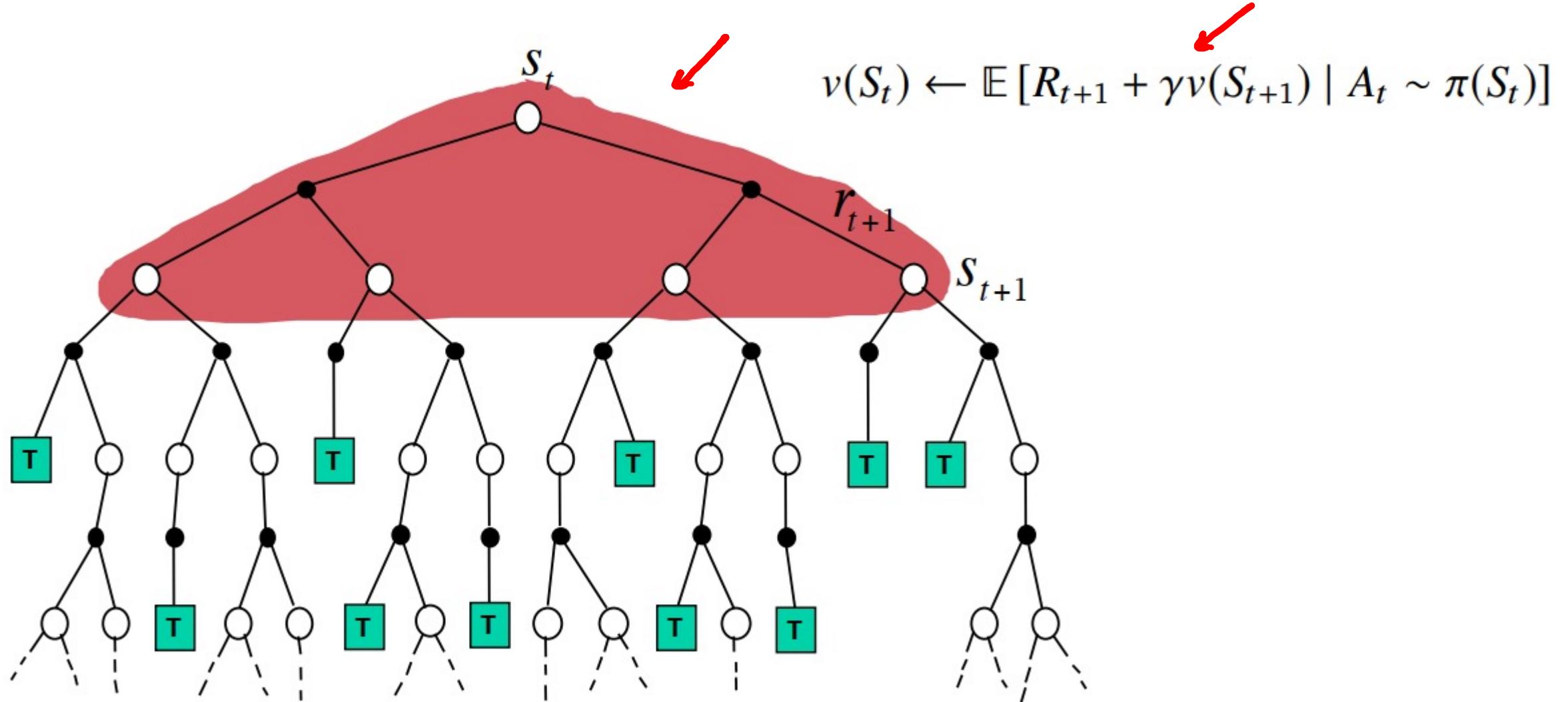
- TD Learning

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

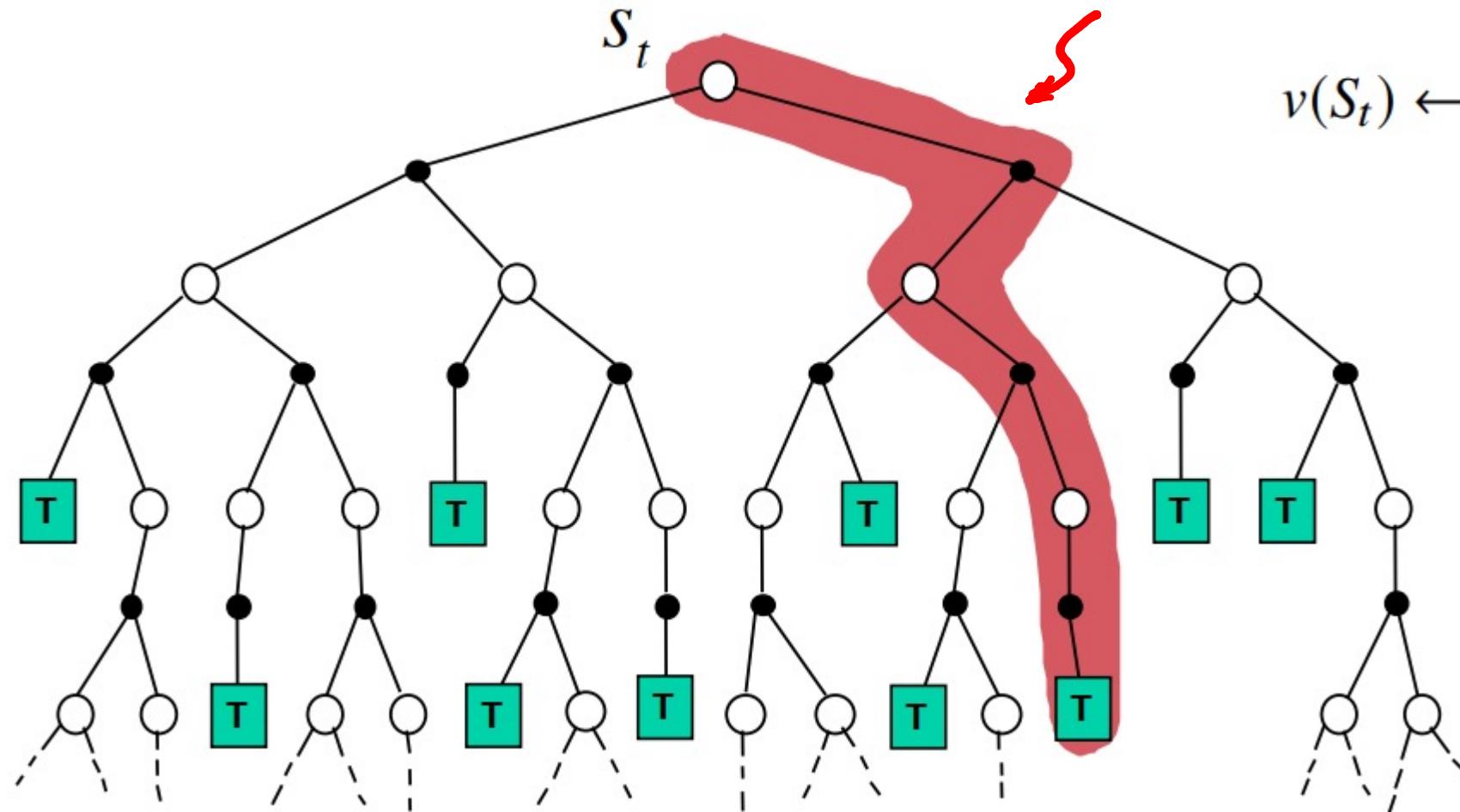
$$v_{t+1}(S_t) \leftarrow v_t(S_t) + \alpha \left(\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t) \right)$$

TD error

Backup (Dynamic Programming)

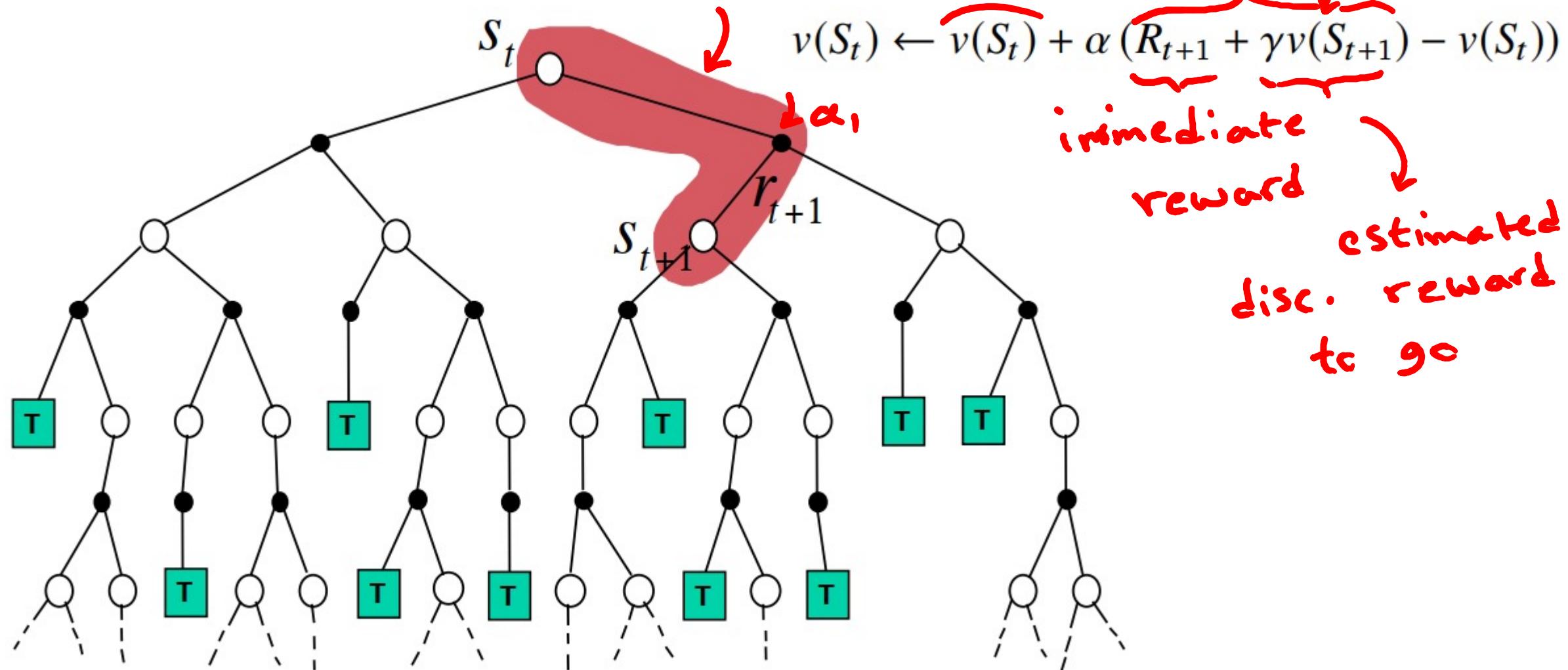


Backup (Monte Carlo)



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

Backup (Temporal Difference)



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(1)
↓

TD Learning for action values

$$S_0, A_0, R_1, S_1, A_1, \dots \sim \pi_t$$

TD(0)

- We can apply the same idea to action values
- Temporal-difference learning for action values:

SARSA

- Update value $q_t(S_t, A_t)$ towards estimated return $R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$

$$\sum \alpha_t = \infty$$

$$\sum \alpha_t^2 < \infty$$

$$\alpha \rightarrow 0$$

$$q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha$$

$$q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha \left(\underbrace{R_{t+1} + \gamma \underline{q_t(S_{t+1}, A_{t+1})}}_{\text{target}} - \underbrace{q_t(S_t, A_t)}_{\text{Current est.}} \right)$$

TD error

Sample

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 $G_t = R_{t+1} + \gamma R_{t+2} + \dots$ 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:

MC ↗

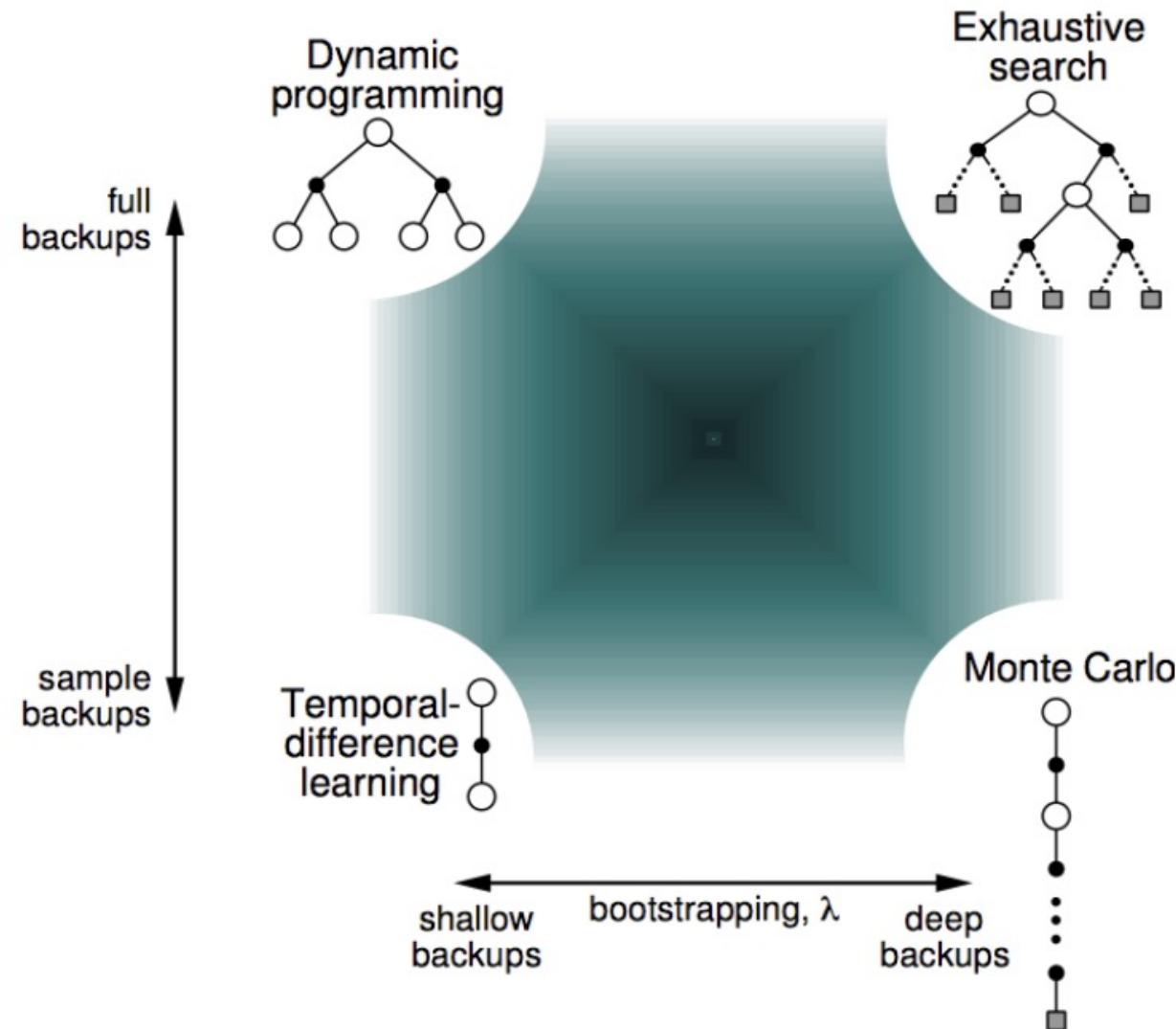
- Return depends on many random actions, transitions, rewards
- TD target depends on one random action, transition, reward

TD ↗

$\text{TD}(1)$ $\text{TD}(0)$

Between MC and TD:
Multi-step TD

General View of Reinforcement Learning



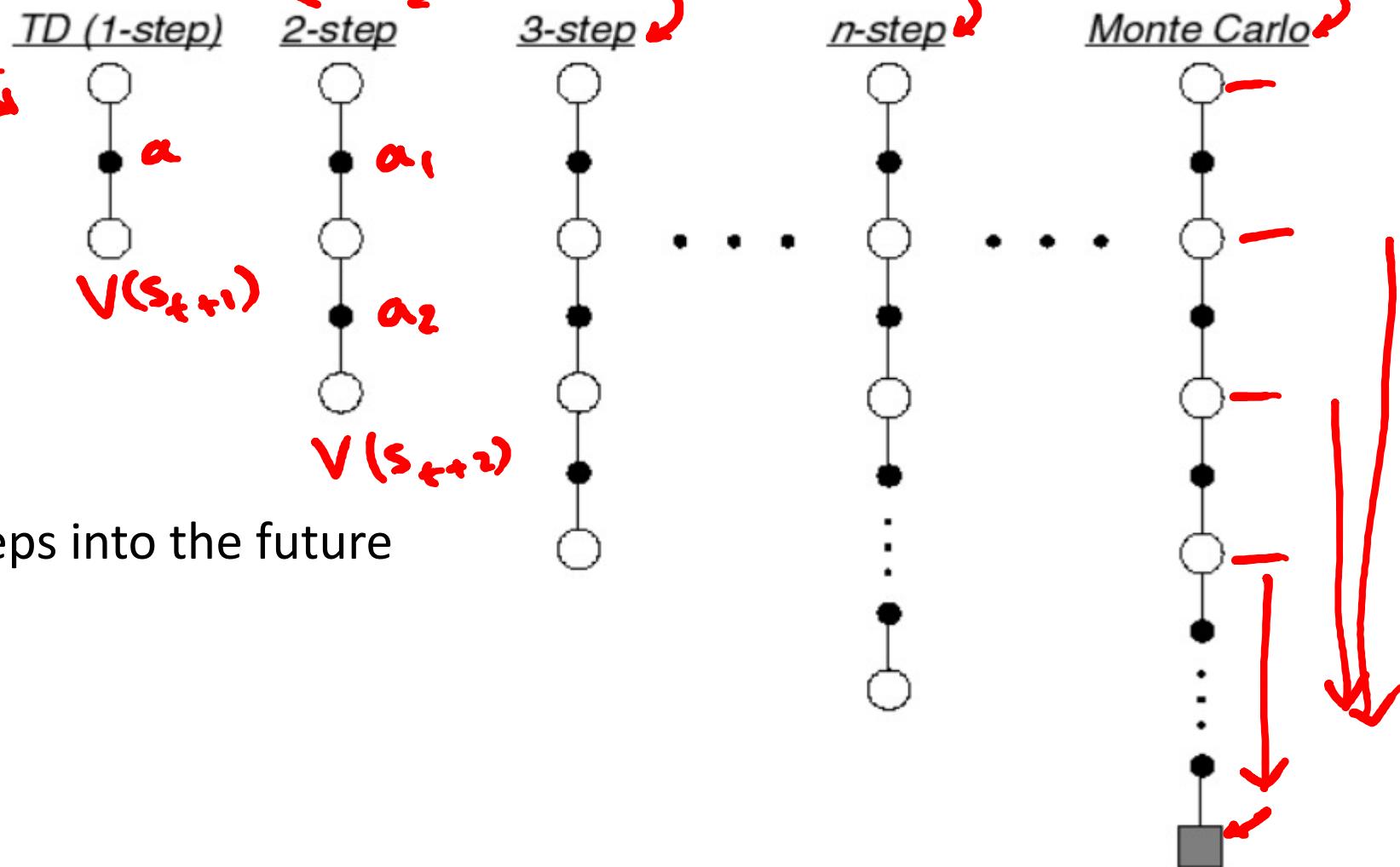
Motivation

- 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

$$R_{t+1} + \gamma V(s_{t+1}) \leftarrow \dots$$

Sample



- Let TD target look n steps into the future

n-Step Returns

- Consider the following n-step returns for $n = 1; 2; \dots$:

$n = 1 \quad (\text{TD}) \quad G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1})$

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

\vdots

$n = \infty \quad (\text{MC}) \quad G_t^{(\infty)} = \underbrace{R_{t+1}} + \underbrace{\gamma R_{t+2}} + \dots + \underbrace{\gamma^{T-t-1} R_T}$

*from time t onwards
& expanding only one action*

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

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

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

$\text{TD}(\alpha)$

$$\overbrace{\text{TD}(\alpha)}^G = (1 - \lambda) \sum_{n=1}^{\infty} \underbrace{\lambda^{n-1}}_i G_t^{(n)}$$

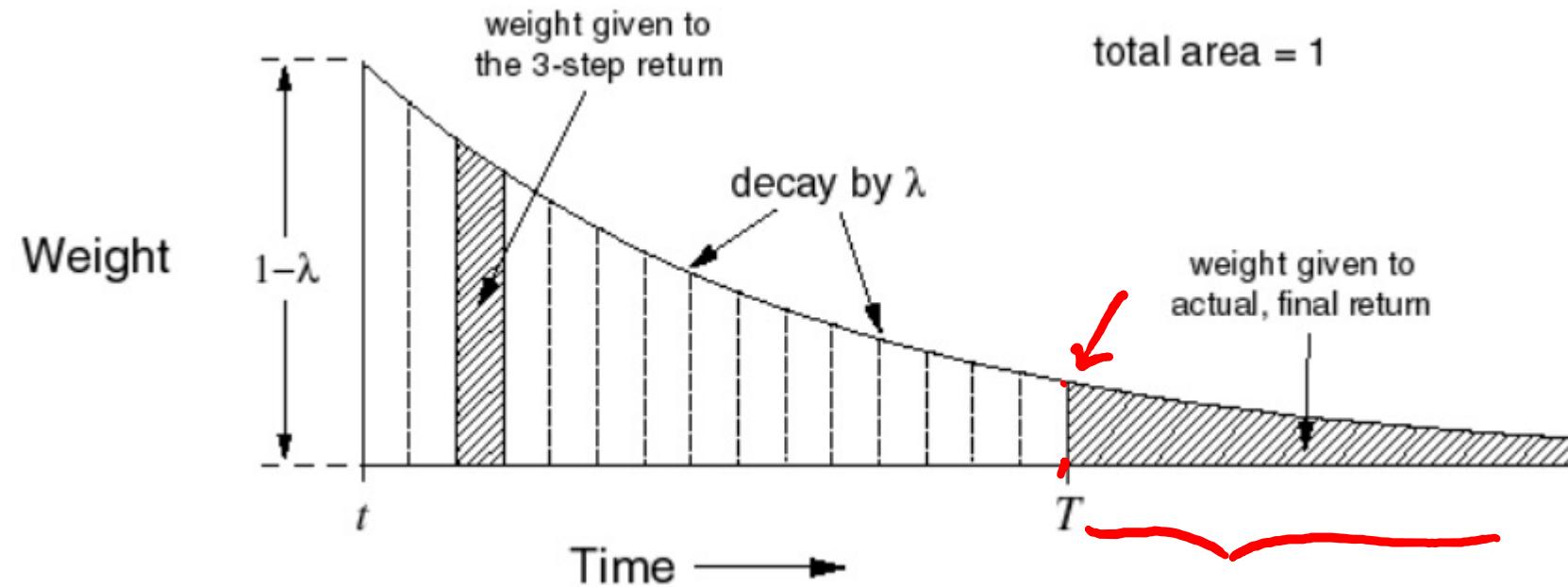
$$0 \leq \lambda \leq 1$$

$$\sum_{i=0}^{\infty} \lambda^i = \frac{1}{1-\lambda}$$

- Forward view $\underline{\text{TD}(\lambda)}$

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

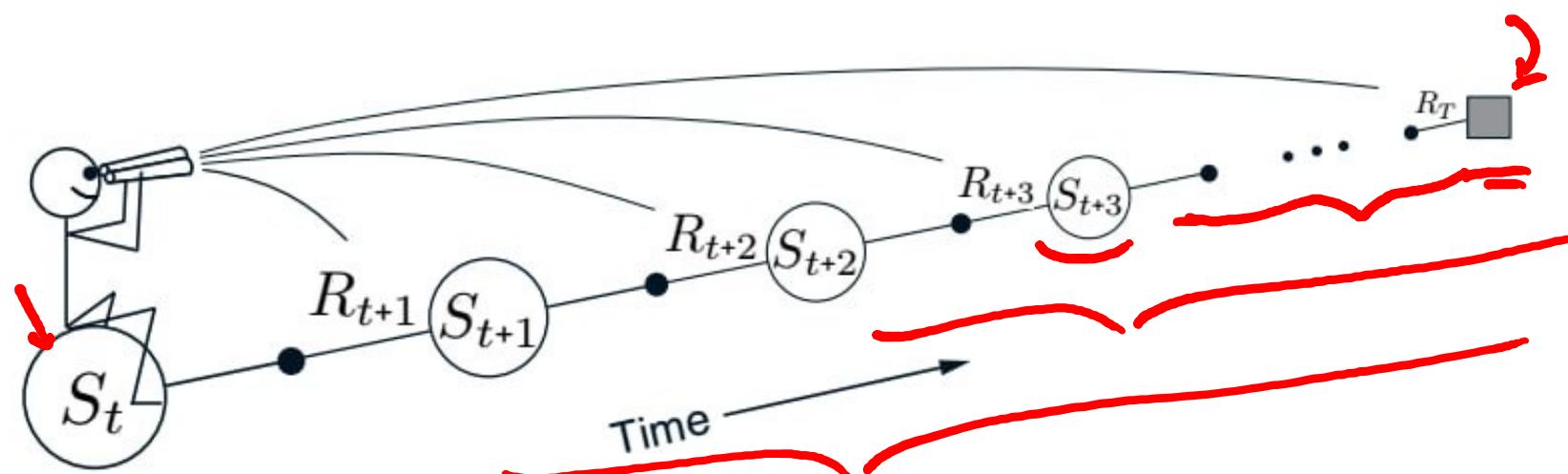
TD(λ) Weighting Function



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

Forward-view TD(λ)

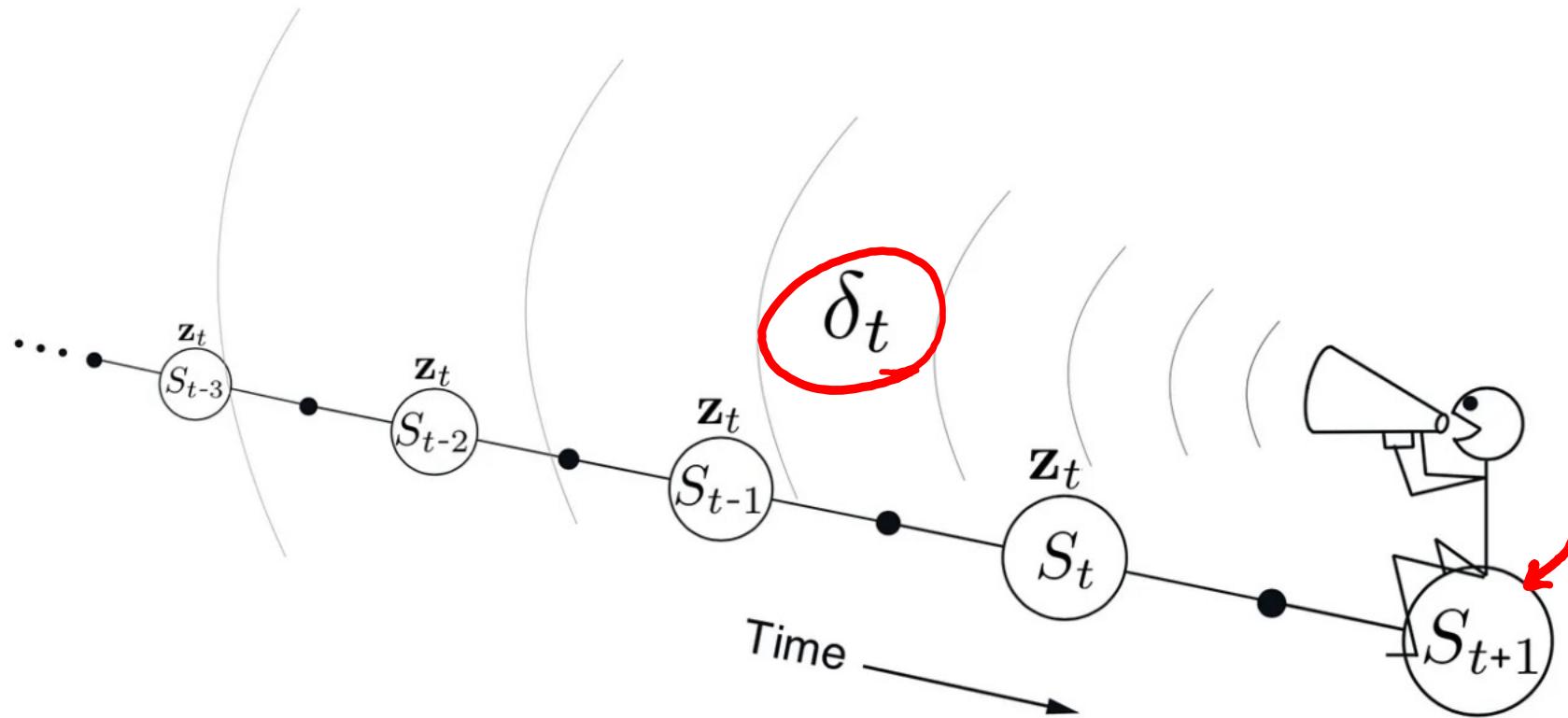
- Update value function towards the λ -return
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes



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

$$E_0(s) = 0$$
$$E_t(s) = \underbrace{\gamma \lambda E_{t-1}(s)}_{\text{Discounted previous trace}} + \underbrace{\mathbf{1}(S_t = s)}_{\text{Reward indicator}}$$

$$\underline{E}_t = \left[\begin{array}{c} \vdots \\ \vdots \end{array} \right]$$

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

Original TD

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(s) \leftarrow V(s) + \underbrace{\alpha \delta_t E_t(s)}_{\text{TD error scaled by trace}}$$

TD(λ) and TD(0)

- When $\lambda = 0$, 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

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

Why Eligibility Traces?

$$G_i^{(t)}$$

$$G_i^{\lambda}$$

$$\begin{aligned}
 \mathcal{T}_\lambda^\pi V(x_0) &= (1 - \lambda) \mathbb{E} \left[\sum_{t \geq 0} \lambda^t \left(\sum_{i=0}^t \gamma^i r^\pi(x_i) + \gamma^{t+1} V(x_{t+1}) \right) \right] \\
 &= \mathbb{E} \left[(1 - \lambda) \sum_{i \geq 0} \underbrace{\gamma^i r^\pi(x_i)}_{\text{Reward}} \sum_{t \geq i} \lambda^t + \sum_{t \geq 0} \underbrace{\gamma^{t+1} V(x_{t+1})}_{\text{Future Value}} (\lambda^t - \lambda^{t+1}) \right] \\
 &= \mathbb{E} \left[\sum_{i \geq 0} \underbrace{\lambda^i}_{\text{Eligibility Trace}} \left(\underbrace{\gamma^i r^\pi(x_i)}_{\text{Reward}} + \gamma^{i+1} V(x_{i+1}) - \underbrace{\gamma^i V(x_i)}_{\text{Current Value}} \right) \right] + V_n(x_0) \\
 &= \mathbb{E} \left[\sum_{i \geq 0} (\gamma \lambda)^i d_i \right] + V(x_0),
 \end{aligned}$$

$\sum_{t \geq i} \lambda^t \Rightarrow \sum_{t \geq i} \frac{\lambda^t}{1-\lambda}$

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

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t^\lambda - V(S_t))$$

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	$\text{TD}(0)$ 	$\text{TD}(\lambda)$ 	$\text{TD}(1)$
Forward view	$\text{TD}(0)$	Forward $\text{TD}(\lambda)$	MC
Online updates	$\lambda = 0$	$\lambda \in (0, 1)$	$\lambda = 1$
Backward view	$\text{TD}(0)$ 	$\text{TD}(\lambda)$ ✗	$\text{TD}(1)$ ✗
Forward view	$\text{TD}(0)$ 	Forward $\text{TD}(\lambda)$ 	MC
Exact Online	$\text{TD}(0)$	Exact Online $\text{TD}(\lambda)$	Exact Online $\text{TD}(1)$

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

Temporal Difference Learning

Control

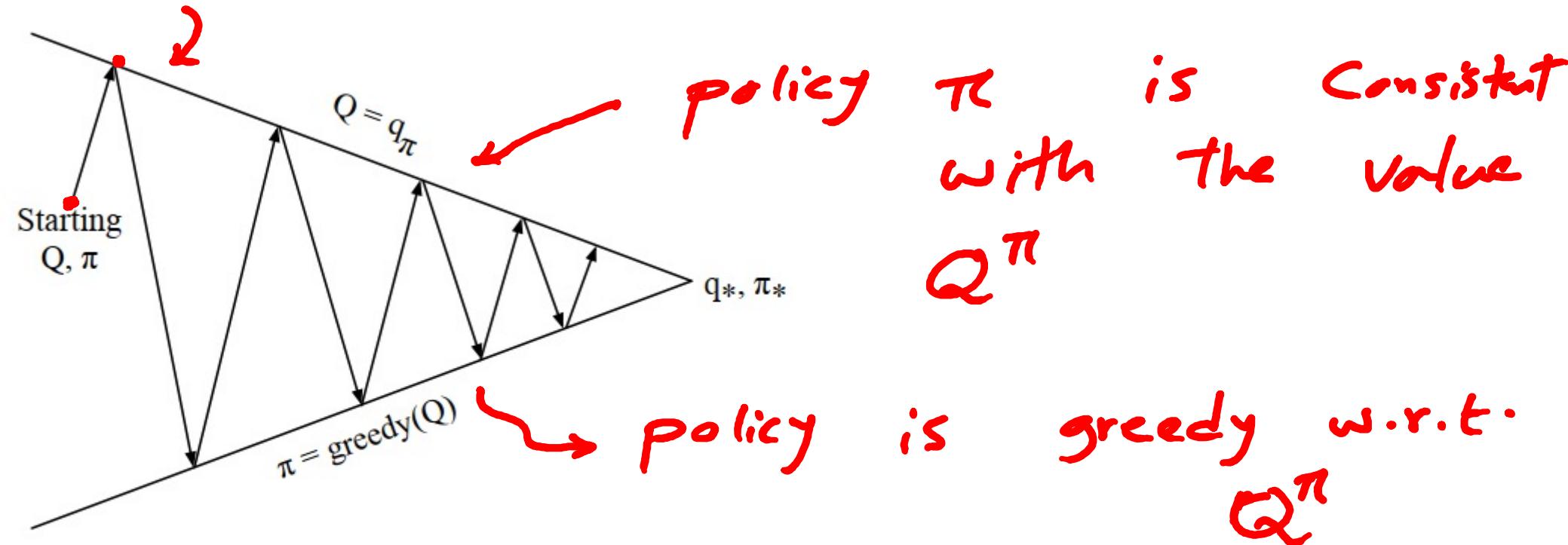
On and Off-Policy Learning

$$\pi_b = \pi_e$$

- 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

$$\pi_b \neq \pi_e$$

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 $1 - \epsilon$ 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, $v_{\pi'}(s) \geq v_\pi(s)$

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

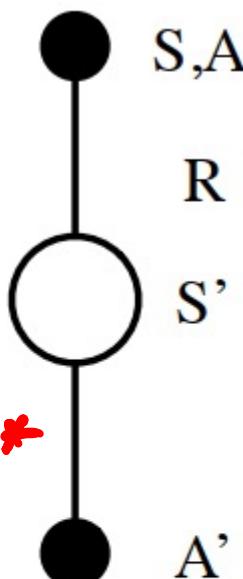
Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

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

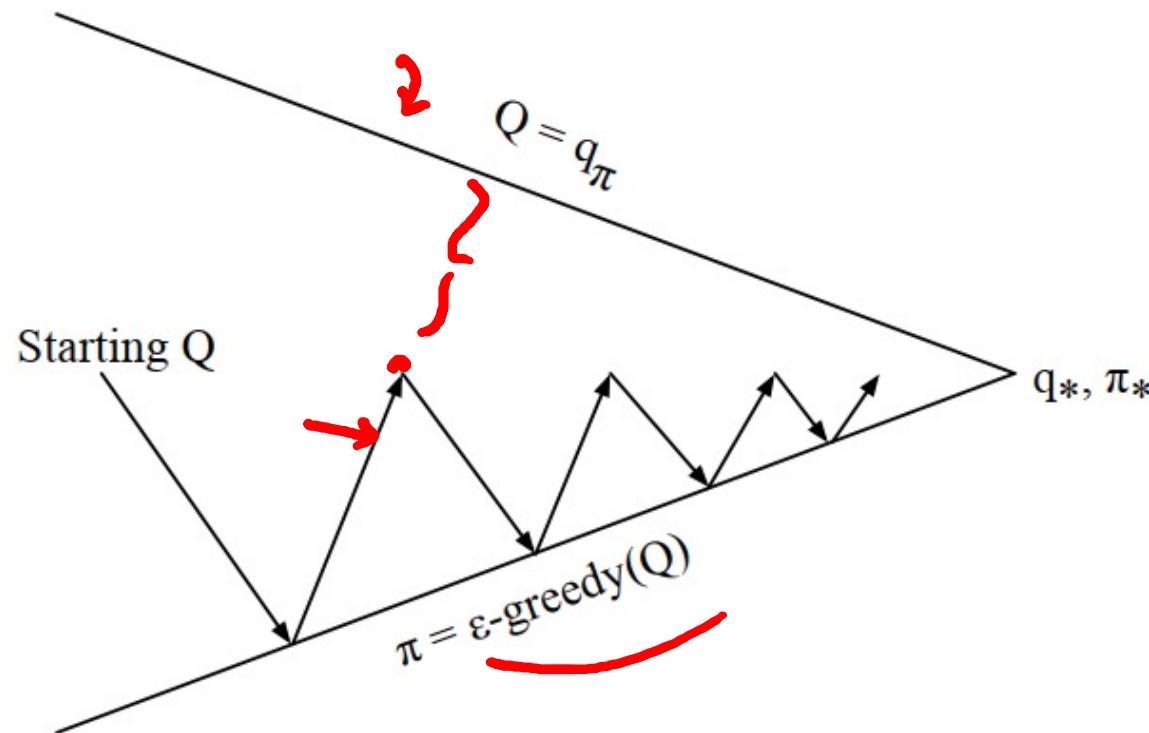
I want to
estimate $Q^*(s, a)$
by using some
random policy $\pi \neq \pi^*$
for sampling episodes.



$$Q^*(s, a) = \mathbb{E}_{\substack{s \\ a}} \left[R(s) + \gamma \max_{a'} Q^*(s', a') \right]$$

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

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

SARSARSA

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

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

Sample

↳ 1-step

n-Step SARSA

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

$$n = 1 \quad (\text{Sarsa}) \quad q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1})$$

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

⋮

$$n = \infty \quad (\text{MC}) \quad q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

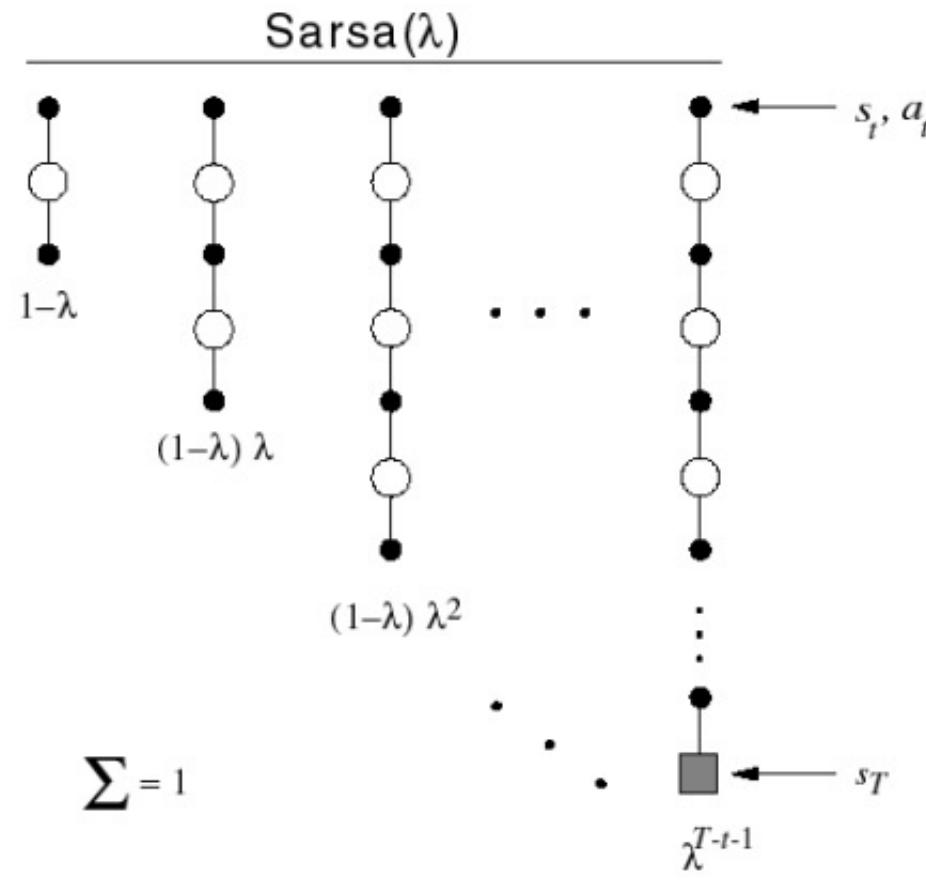
Define the n -step Q-return

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

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)

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

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?” \Rightarrow 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, $q \rightarrow q^$, 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 $Q(\text{terminal-state}, \cdot) = 0$

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'

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

$S \leftarrow S'$;

 until S is terminal