113-1 (Fall 2024) Semester

Reinforcement Learning

Lecture 4:
Model-Free Prediction

Slides

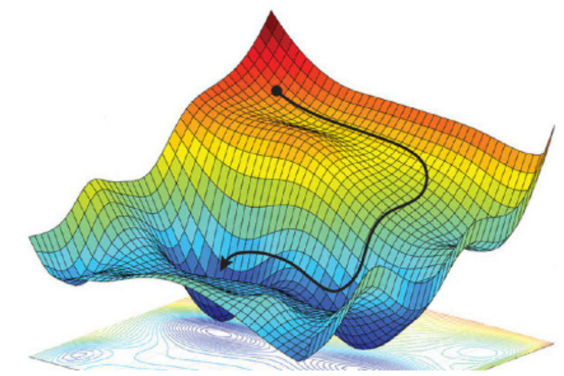
Shao-Hua Sun (孫紹華)

Assistant Professor in Electrical Engineering,
National Taiwan University

Disclaimer

Experimental course

- Everything is subject to change
- Most materials are made from scratch for this course
  - There could be mistakes and flaws in slides, assignments, etc.
- We are altogether in an early iteration of gradient descent
  - TAs and I are the optimizer, you (and your feedback) are the data



[source]

Credit to David Silver

- This part of the course — Introduction to Reinforcement Learning — is 99% based on David Silver's Reinforcement Learning lecture at University College London with DeepMind
- You are highly encouraged to watch David's lectures

Schedule

Week	Date	Topic	Assignment	Project
1	9/5	• Lecture 0: Course Introduction • Lecture 1: Introduction to Reinforcement Learning		
2	9/12	• Lecture 2: Markov Decision Processes • Lecture 3: Planning by Dynamic Programming	#1 Release	
3	9/19	• Lecture 4: Model-Free Prediction • Lecture 5: Model-Free Control		
4	9/26	• Lecture 6: Value Function Approximation • Lecture 7: Policy Gradient Methods	#1 Due #2 Release	

- Lecture 4 Model-Free Prediction (9:30 AM-10:50 AM)
- Lecture 5 Model-Free Control (11 AM - 12:10 PM)

Schedule

Week	Date	Topic	Assignment	Project
1	9/5	• Lecture 0: Course Introduction • Lecture 1: Introduction to Reinforcement Learning		
2	9/12	• Lecture 2: Markov Decision Processes • Lecture 3: Planning by Dynamic Programming	#1 Release	
3	9/19	• Lecture 4: Model-Free Prediction • Lecture 5: Model-Free Control		
4	9/26	• Lecture 6: Value Function Approximation • Lecture 7: Policy Gradient Methods	#1 Due #2 Release	

- Assignment #1
  - **Deadline:** 9/26 9:30 AM (no late submission)
- **Assignment #2 TA session** (9/26 11:40 AM - 12:10 PM) by 楊可 Co Yong
  - **Release:** 9/26 12:10 PM on NTU COOL
  - **Deadline:** 10/17 9:30 AM (no late submission)

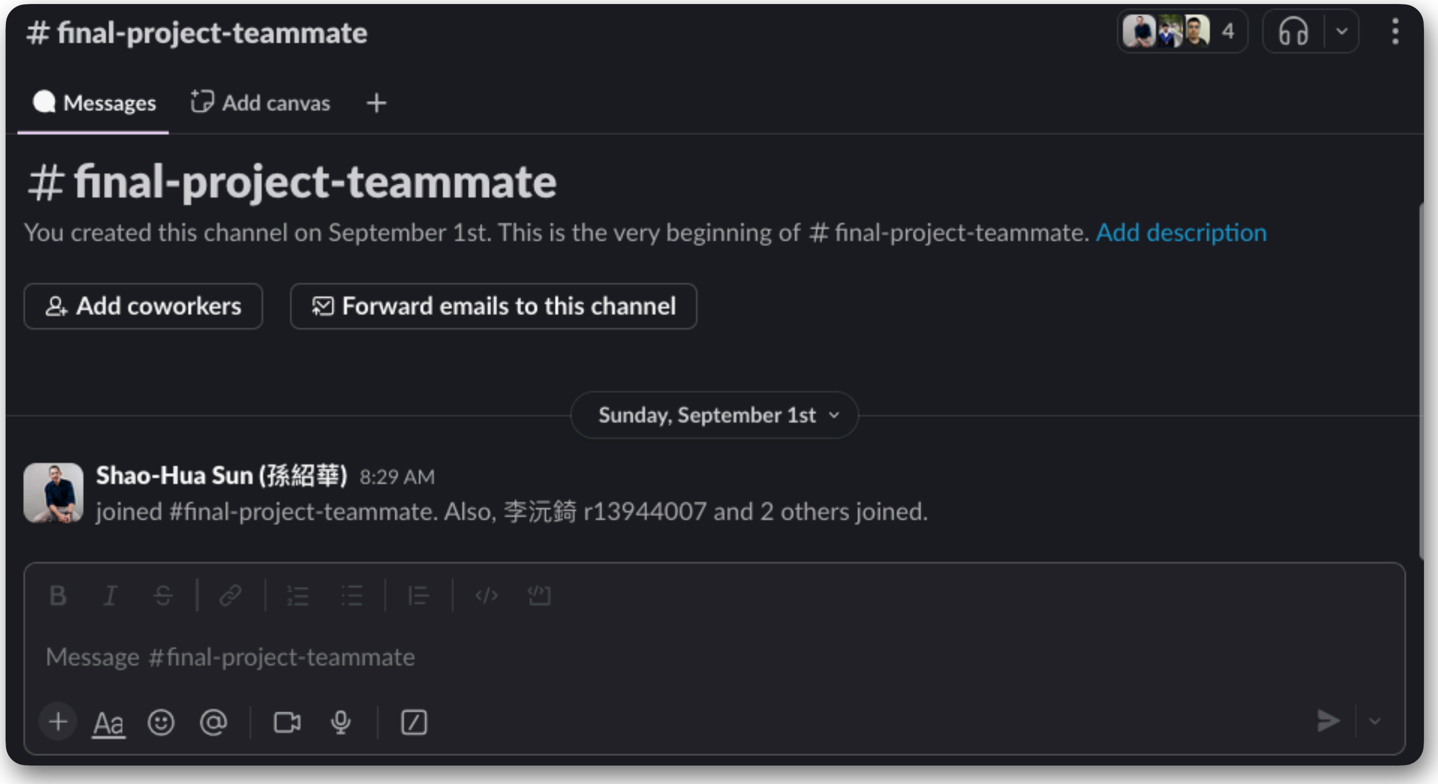
Final Project - Form Your Team

Form a team with 4 members

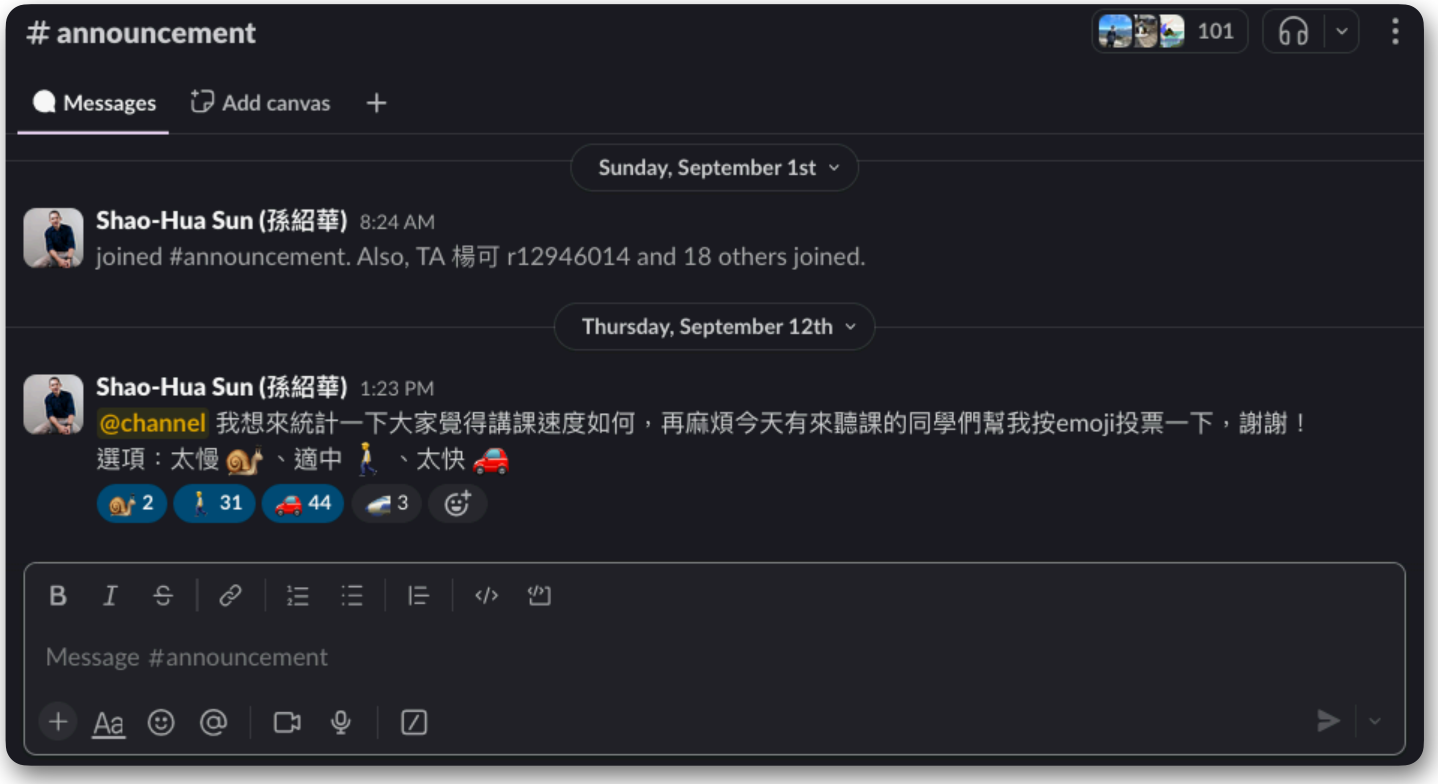
- Friends
- Lab mates
- Find them on slack #final-project-teammate

Based on

- Research interests
- Work habits



The Pace of Lectures



Week	Date	Topic	Assignment	Project
1	9/5	• Lecture 0: Course Introduction • Lecture 1: Introduction to Reinforcement Learning		
2	9/12	• Lecture 2: Markov Decision Processes • Lecture 3: Planning by Dynamic Programming	#1 Release	
3	9/19	• Lecture 4: Model-Free Prediction • Lecture 5: Model-Free Control		
4	9/26	• Lecture 6: Value Function Approximation • Lecture 7: Policy Gradient Methods	#1 Due #2 Release	
5	10/3	• Lecture 8: Integrating Learning and Planning • Lecture 9: Exploration and Exploitation		
6	10/10	• Lecture 10: Deep Q-Learning • Lecture 11: Deep Policy Optimization		
7	10/17	• Lecture 11: Deep Policy Optimization • Lecture 12: Deep Q-Learning + Policy Optimization	#2 Due #3 Release	
8	10/24	• Lecture 13: Imitation Learning • Lecture 14: Skill-based RL • Lecture 15: Offline RL		

Week	Date	Topic	Assignment	Project
9	10/31	• Lecture 16: Multi-task RL • Lecture 17: Meta RL • Lecture 18: Hierarchical RL		Confirm team members and potential topics
10	11/7	• Lecture 19: RL Exploration • Lecture 20: Model-based RL • Lecture 21: Programmatic RL • Lecture 22: RL from Human Feedback	#3 Due	
11	11/14	• Final Project Proposal		Meet with TA
12	11/21	• Jiayuan Mao (MIT) • Karl Pertsch (UC Berkeley & Stanford)		Meet with the instructor
13	11/28	• Youngwoon Lee (UC Berkeley) • Guanzhi Wang (Caltech & Nvidia)		Meet with TA
14	12/5	• Risto Vuorio (University of Oxford) • Kuang-Huei Lee (Google DeepMind)		Meet with the instructor
15	12/12	• Aleksei Petrenko (Apple) • Ping-Chun Hsieh (NYCU)		Meet with TA
16	12/19	• Final Project Presentation		Report deadline (12/22 11:59 PM)

Recap

Markov Process and its Variants

Category	Reward	Action	Problem
Markov Process (Markov Chain)	❌	❌	
Markov Reward Process (MRP)	✅	❌	Prediction
Markov Decision Process (MDP)	✅	✅	Prediction & Control

Bellman Equation - Summary

Bellman expectation equations

- State-value function v_π

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')\right)$$

- Action-value function q_π

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s',a')$$

Bellman optimality equations

- Optimal state-value function v_*

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

- Optimal action-value function q_*

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a')$$

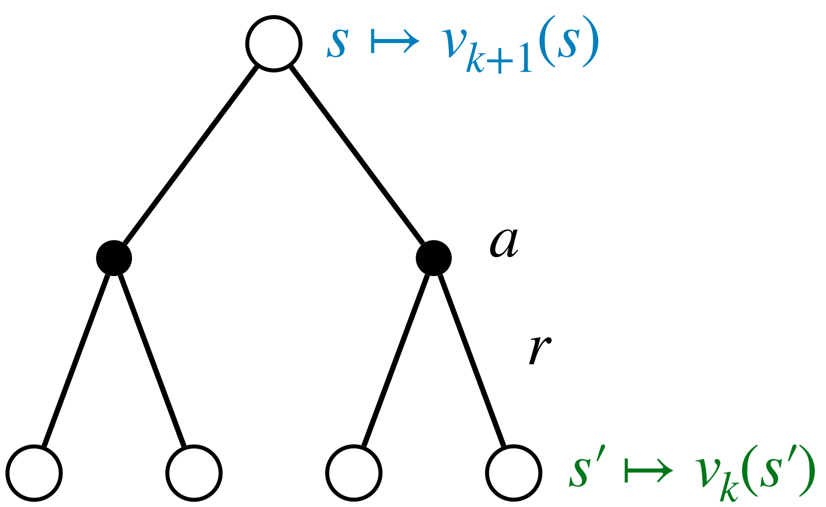
Iterative Policy Evaluation

Policy evaluation

- **Problem**: evaluate a given policy π
- **Solution**: iteratively apply Bellman expectation backup
  - v_1 (arbitrarily initialized) $\rightarrow v_2 \rightarrow v_3 \rightarrow \ldots \rightarrow v_\pi$

Policy evaluation procedure

- At each iteration $k+1$
- For all states $s \in S$
- Update $v_{k+1}(s)$ from $v_k(s')$ by



$$v_{k+1}(s) = \sum_{a \in A} \pi(a \mid s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')\right) \quad \text{or} \quad v_{k+1} = R^\pi + \gamma P^\pi v_k$$

Convergence proof by the contraction mapping theorem: reference

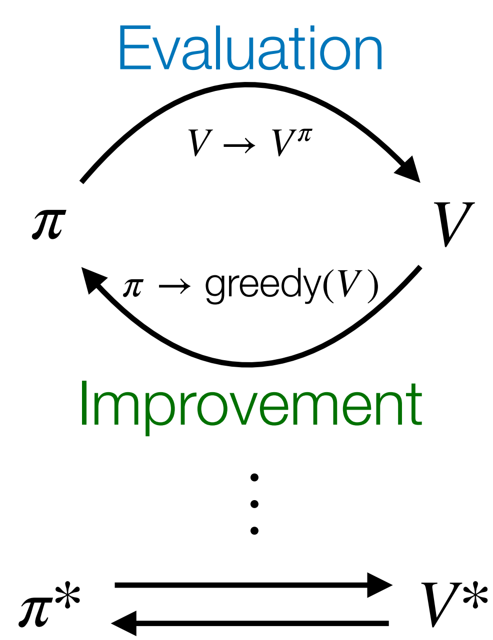
Policy Iteration

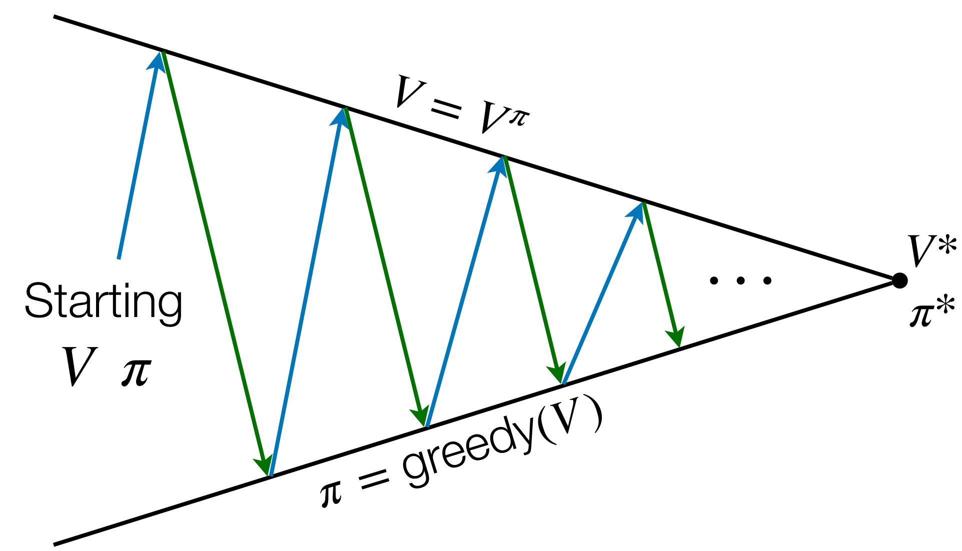
Given policy π

- Evaluate the policy
- Improve the policy by acting greedily with respect to v_π, $\pi' = \text{greedy}(v_\pi)$

Policy iteration

- Policy evaluation
  - Estimate v_π
  - Iterative policy evaluation
- Policy improvement
  - Generate $\pi' \geq \pi$
  - Greedy policy improvement



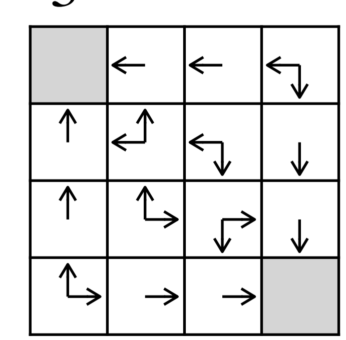


Gridworld: the improved policy was optimal, $\pi' = \pi_*$ when $k = 3$

- In general, it needs more iterations of **improvement / evaluation**
  - This process of policy iteration always converges to π_*

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

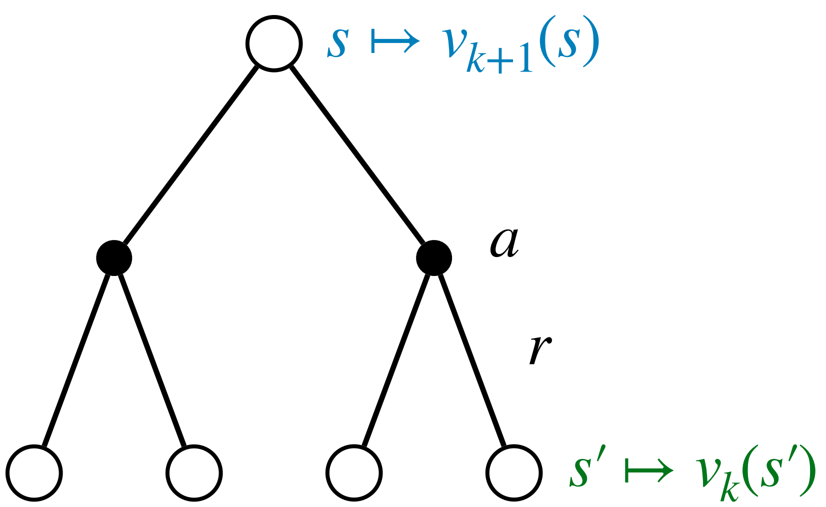


Value Iteration

Value iteration

- **Problem**: find an optimal policy π
- **Solution**: iteratively apply Bellman optimality backup
  - v_1 (arbitrarily initialized) $\rightarrow v_2 \rightarrow v_3 \rightarrow \ldots \rightarrow v_*$ (c.f., v_π in iterative policy evaluation)

Value iteration procedure



- At each iteration $k+1$
- For all states $s \in S$
- Update $v_{k+1}(s)$ from $v_k(s')$ by

$$v_{k+1}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right) \quad \text{or} \quad v_{k+1} = \max_{a \in A} R^a + \gamma P^a v_k$$

Convergence proof by the contraction mapping theorem: reference

Unlike **policy iteration**, there is no explicit policy

- Intermediate value functions may not correspond to any policy

Synchronous Dynamic Programming

Problem	Bellman Equation	Algorithm
Prediction	Bellman Exception Equation	Iterative Policy Evaluation
Control	Bellman Exception Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Complexity

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
  - Complexity $O(mn^2)$ per iteration for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
  - Complexity $O(m^2n^2)$ per iteration

Asynchronous Dynamic Programming

Synchronous DP backups

- All states are backed up in parallel

v_1

0	0	0
0	0	0
0	0	0

$\Rightarrow$ v_2

0	-1	-1
-1	-1	-1
-1	-1	-1

Asynchronous DP backups

- Backs up states individually, in any order
- For each selected state, apply the appropriate backup
- Can significantly reduce **computation**
- **Convergence**: guaranteed to converge if all states continue to be selected

$v_1(s_1)$

0	$\boxed{0}$	0
0	0	0
0	0	0

$\Rightarrow$ $v_2(s_1)$

0	$\boxed{-1}$	0
0	0	0
0	0	0

Ideas for asynchronous DP backups

- *In-place* dynamic programming
- *Prioritized* sweeping
- *Real-time* dynamic programming

Outline

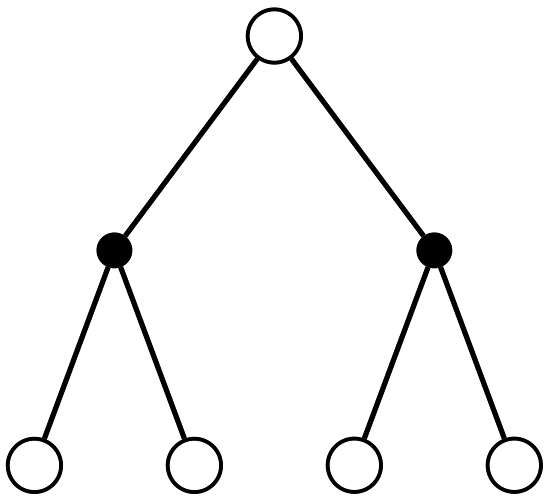
- Introduction
- Monte-Carlo Learning
- Temporal-Difference Learning
- TD(λ)

Outline

- Introduction
- Monte-Carlo Learning
- Temporal-Difference Learning
- TD(λ)

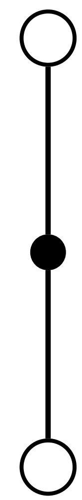
Model-Free Reinforcement Learning

Last lecture (lecture 3)



- **Problem**: solve a known MDP
  - **Prediction**: policy evaluation
  - **Control**: policy iteration and value iteration
- **Solution**: planning by dynamic programming

This lecture (lecture 4)



- **Problem**: estimate the value function of an unknown MDP
  - Model-free prediction

Next lecture (lecture 5)

- **Problem**: optimize the value function and/or policy of an unknown MDP
  - Model-free control

Outline

- Introduction
- Monte-Carlo Learning
- Temporal-Difference Learning
- TD(λ)

Monte-Carlo Learning

Monte-Carlo (MC) reinforcement learning

- MC methods learn directly from episodes of experience
- MC is **model-free**: no knowledge of MDP transitions / rewards
- MC learns from **complete episodes**: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- **Caveat**: can only apply MC to episodic MDPs
    - All episodes must terminate

Monte-Carlo policy evaluation

- **Goal**: learn v_π from episodes of experience under policy π,

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

    - The return is the total discounted reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-1} R_T$
    - The value function is the expected return: $v_\pi(s) = \mathbb{E}[G_t \,|\, S_t = s]$
- Monte-Carlo policy evaluation uses *empirical mean return* instead of expected return

Monte-Carlo Policy Evaluation

First-visit Monte-Carlo policy evaluation

- To evaluate state s, the first time-step t that state s is visited in an episode
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $Returns(s) \leftarrow Returns(s) + G_t$
  - Value is estimated by mean return $V(s) = Returns(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Every-visit Monte-Carlo policy evaluation

- To evaluate state s, every time-step t that state s is visited in an episode
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $Returns(s) \leftarrow Returns(s) + G_t$
  - Value is estimated by mean return $V(s) = Returns(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Example - Blackjack

Blackjack MDP

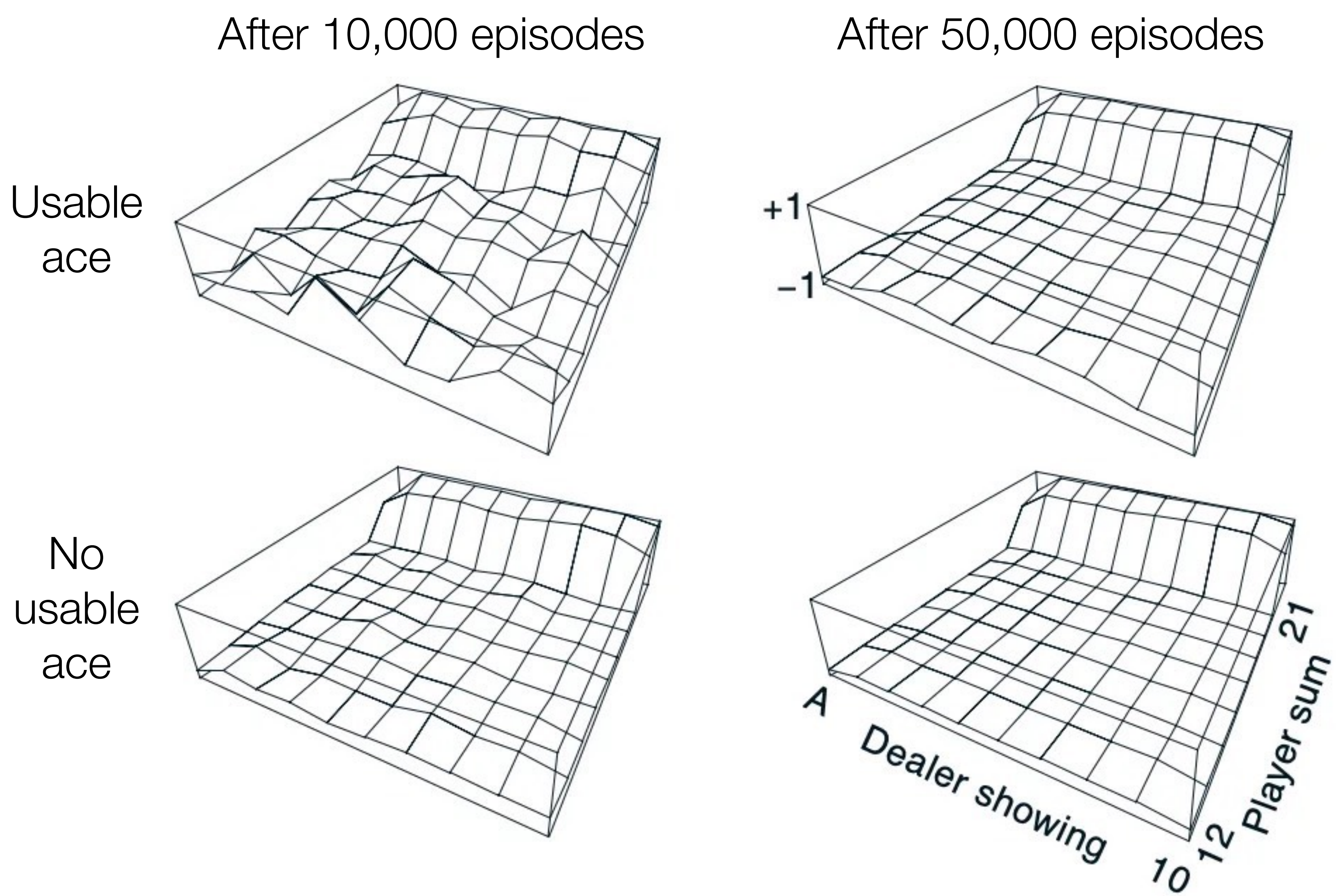
- States (200 of them)
  - Current sum (12-21), dealer's showing card (ace-10), do I have a "useable" ace? (yes/no)
- Actions
  - stand/stick: stop receiving cards (and terminate)
  - hit/twist: take another card (no replacement)
- Reward
  - stand/stick: stop receiving cards (and terminate)
    - +1 if sum of cards > sum of dealer cards
    - 0 if sum of cards = sum of dealer cards
    - -1 if sum of cards < sum of dealer cards
  - hit/twist: take another card (no replacement)
    - -1 if sum of cards > 21 (and terminate), 0 otherwise
- Transitions: automatically hit if sum of cards < 12

[source]

Blackjack Value Function with Monte-Carlo

Blackjack MDP

- **Policy:** stand if sum of cards ≥ 20, otherwise hit



Incremental Monte-Carlo Updates

Incremental mean

- The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally,

$$\mu_k = \frac{1}{k}\sum_{t=1}^{k} x_t = \frac{1}{k}\left(x_k + \sum_{t=1}^{k-1} x_t\right) = \frac{1}{k}[x_k + (k-1)\mu_{k-1}] = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Incremental Monte-Carlo updates

- Update $V(s)$ incrementally after episodes $S_1, A_1, R_2, \ldots, S_T$
- For each state S_t with return G_t
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Update the value $V(S_t) \leftarrow V(S_t) + \dfrac{1}{N(S_t)}(G_t - V(S_t))$
- In **non-stationary problems**, it can be useful to track a running mean, i.e., forget old episodes, $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$, where α is the step size

Outline

- Introduction
- Monte-Carlo Learning
- Temporal-Difference Learning
- TD(λ)

Temporal-Difference Learning

Temporal-difference (TD) learning

- 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

MC and TD: both learn v_π from episodes of experience under policy π

- **Incremental every-visit Monte-Carlo**: learn from a complete episode
  - Update value $V(S_t)$ toward *actual* return G_t by $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$
- **Simplest temporal-difference learning algorithm: TD(0)**
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

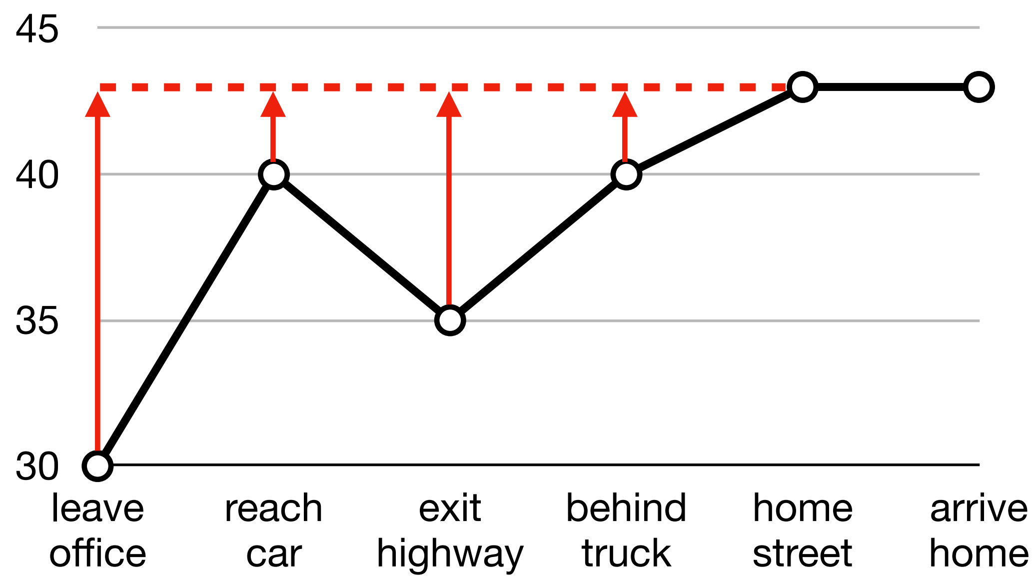
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the **TD target**
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the **TD error**

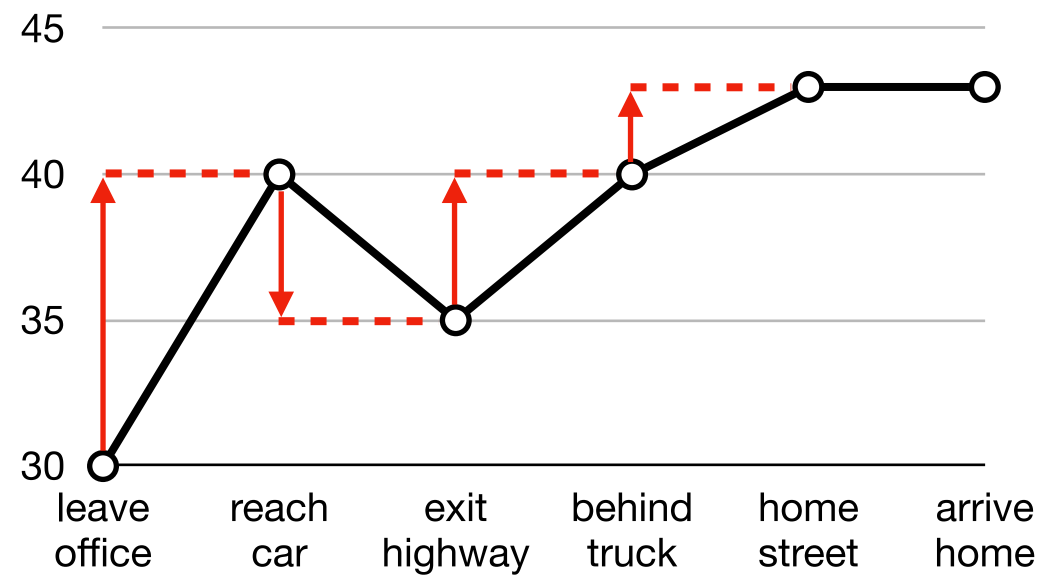
Example - Driving Home

State	Elapsed Time	Predicted Time to Go	Predicted Total Time
Leave office	0	30	30
Reach car, raining	5	35	40
Exit highway	20	15	35
Behind truck	30	10	40
Home street	40	3	43
Arrive home	43	0	43

Changes recommended by **MC** methods



Changes recommended by **TD** methods



MC vs. TD - Final Outcome

MC backup

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

TD backup

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

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

- TD can learn from incomplete sequences
- MC can only learn from complete sequences
- TD works in continuing (non-terminating) environments
- MC only works for episodic (terminating) environments

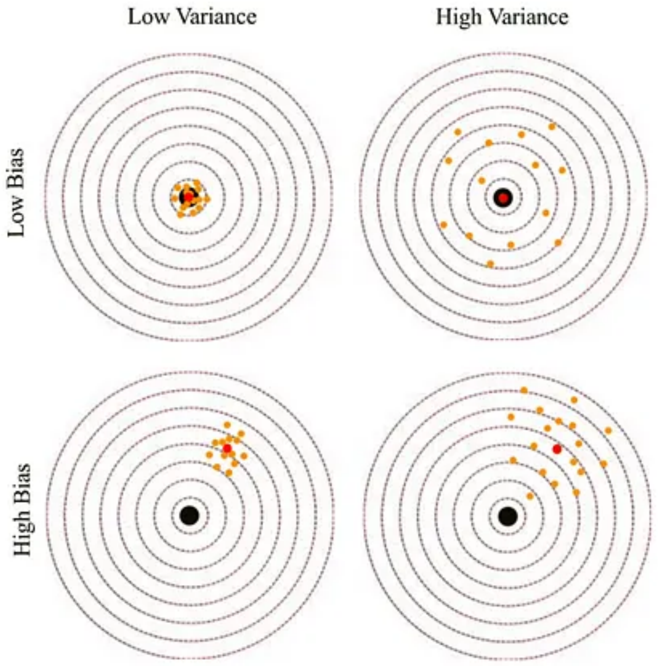
MC vs. TD - Bias/Variance Trade-Off

MC has high variance, zero bias

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(S_t)$
- Good convergence properties (even with function approximation)
- Not very sensitive to initial value
- Very simple to understand and use

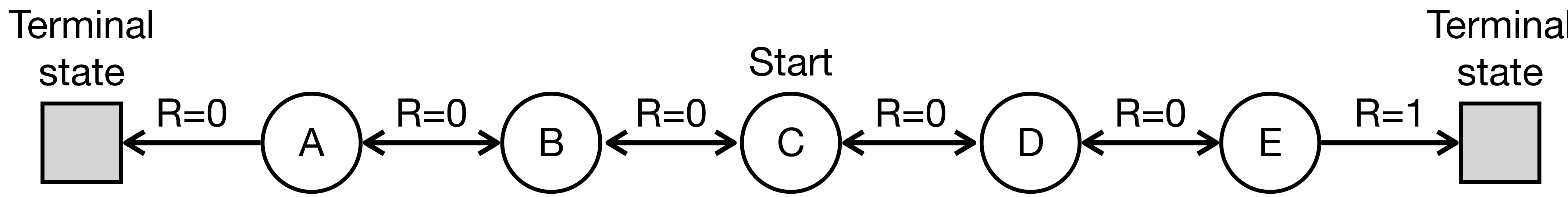
TD has low variance, some bias

- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate of $v_\pi(S_t)$
    - TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return
    - Return depends on many random actions, transitions, rewards
    - TD target depends on one random action, transition, reward
- Usually more efficient than MC
- TD(0) converges to $v_\pi(s)$ (but not always with function approximation)
- More sensitive to initial value

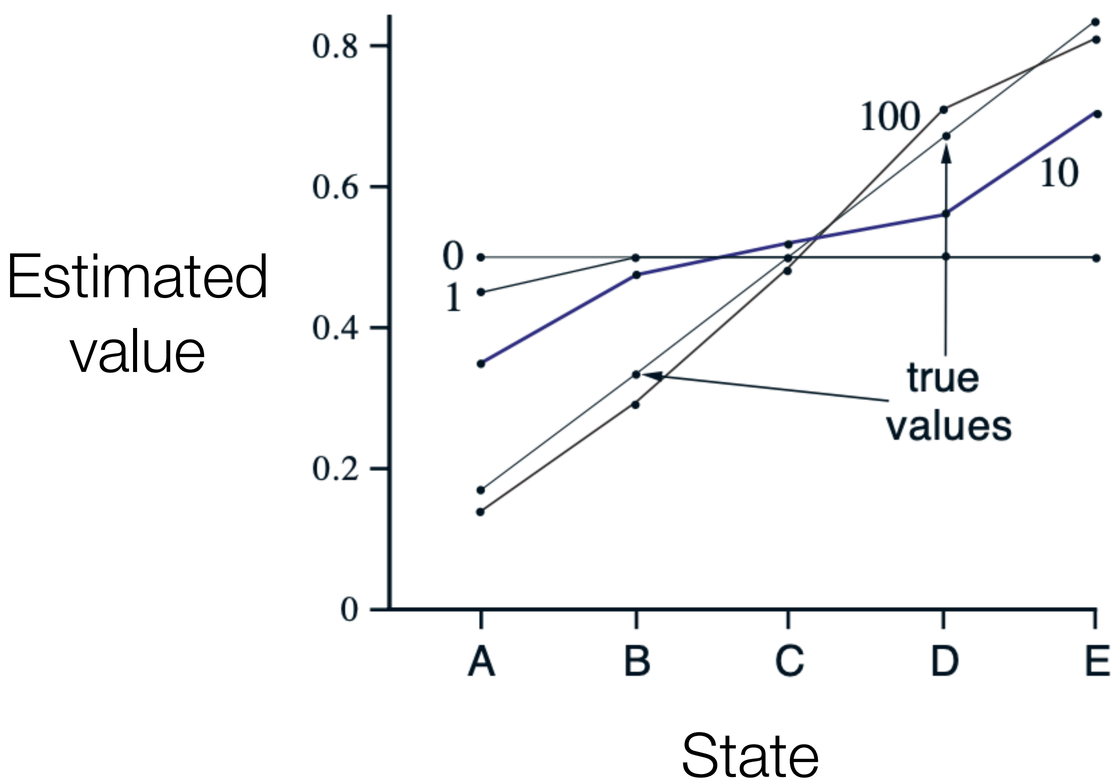


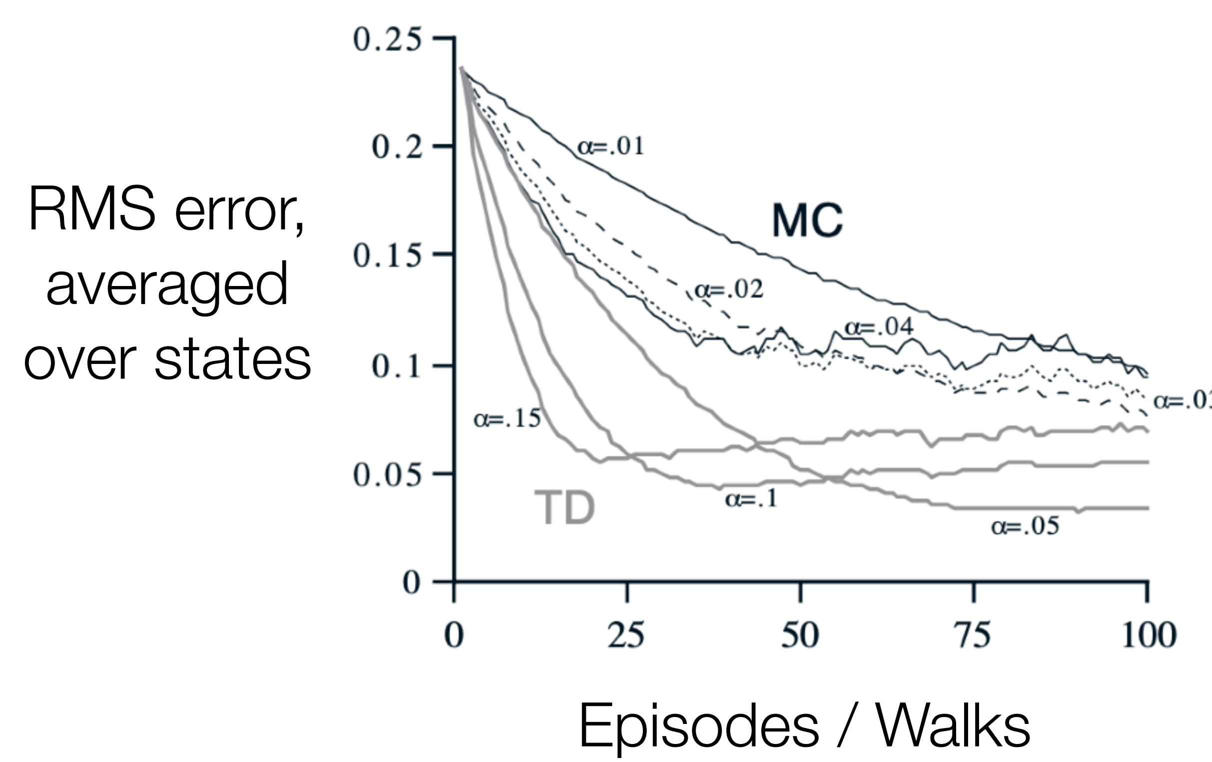
[source]

Example - Random Walk



Policy: agent follows a uniform random policy, $\pi(\rightarrow|\cdot) = \pi(\leftarrow|\cdot) = 0.5$





Batch MC and TD

MC and TD converge: $V(s) \to v_\pi(s)$ as experience $\to \infty$

- But what about batch solution for finite experience?

$$\text{Episode 1} \quad s_1^1, a_1^1, r_2^1, \ldots, s_{T_1}^1$$
$$\vdots$$
$$\text{Episode K} \quad s_1^K, a_1^K, r_2^K, \ldots, s_{T_1}^K$$

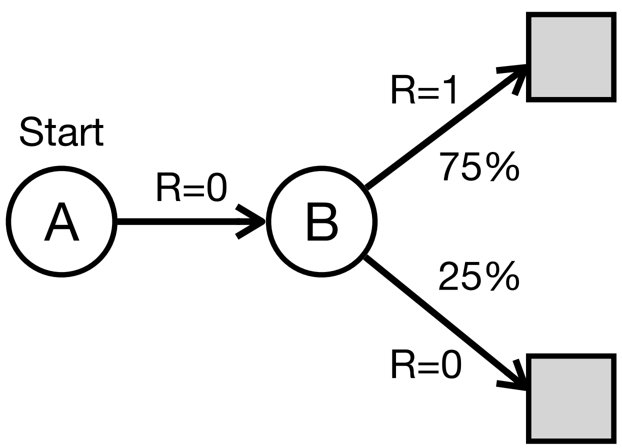
  - Repeatedly sample episode $k \in [1, K]$ and apply MC or TD(0) to episode k

Example: two states A, B; no discounting; 8 episodes of experience; what is $V(A)$, $V(B)$?

- A, 0, B, 0
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 0

Value	TD	MC
V(A)	0.75	0
V(B)	0.75	0.75

Underlying MDP



MC vs. TD - Exploit Markov Property

MC converges to solution with *minimum mean-squared error*

- Best fit to the observed returns

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} (G_t^k - V(s_t^k))^2$$

  - In the AB example, $V(A) = 0$

- **MC does not exploit Markov property:** usually more effective in **non-Markov environments**

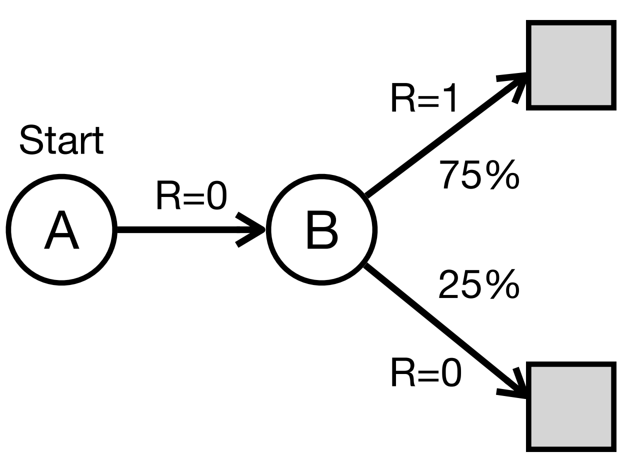
TD(0) converges to solution of *max likelihood Markov model*

- Solution to the MDP $\langle S, A, \hat{P}, \hat{R}, \gamma \rangle$ that best fits the data

$$\hat{P}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\hat{R}^a_s = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k = s, a) r_t^k$$

Underlying MDP

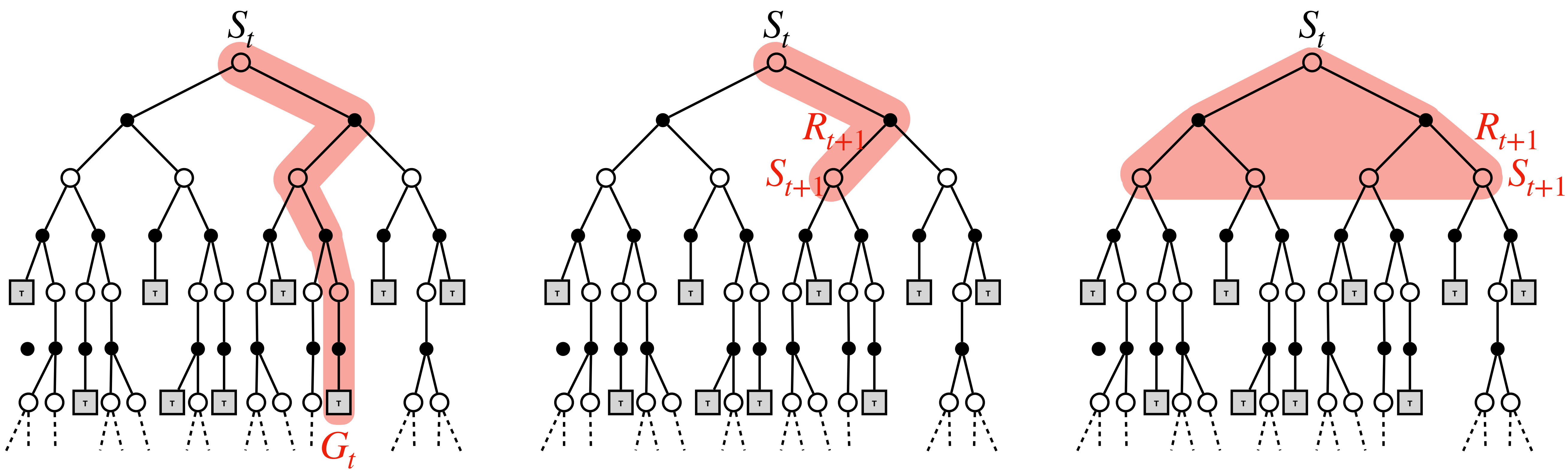


  - In the AB example, $V(A) = 0.75$

- **TD exploits Markov property:** usually more efficient in **Markov environments**

Comparison - MC, TD, and DP

MC backup	TD backup	DP backup
$V(S_t) + \alpha(G_t - V(S_t))$	$V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$	$\mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$



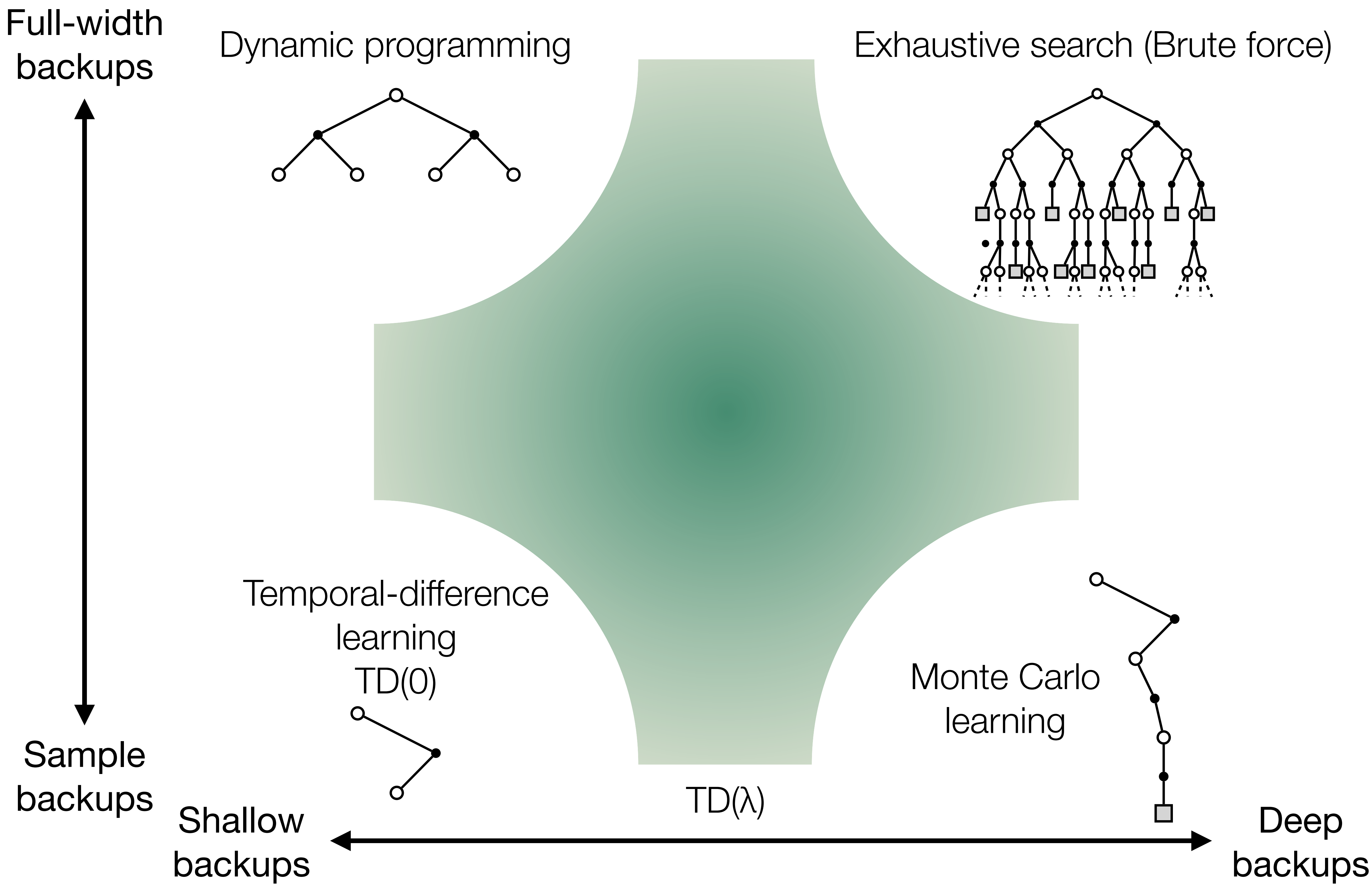
Bootstrapping: update involves an estimate

- **MC** does not bootstrap; **TD** and **DP** bootstrap

Sampling: update samples an expectation

- **MC** and **TD** sample; **DP** does not sample

Comparison - MC, TD, and DP

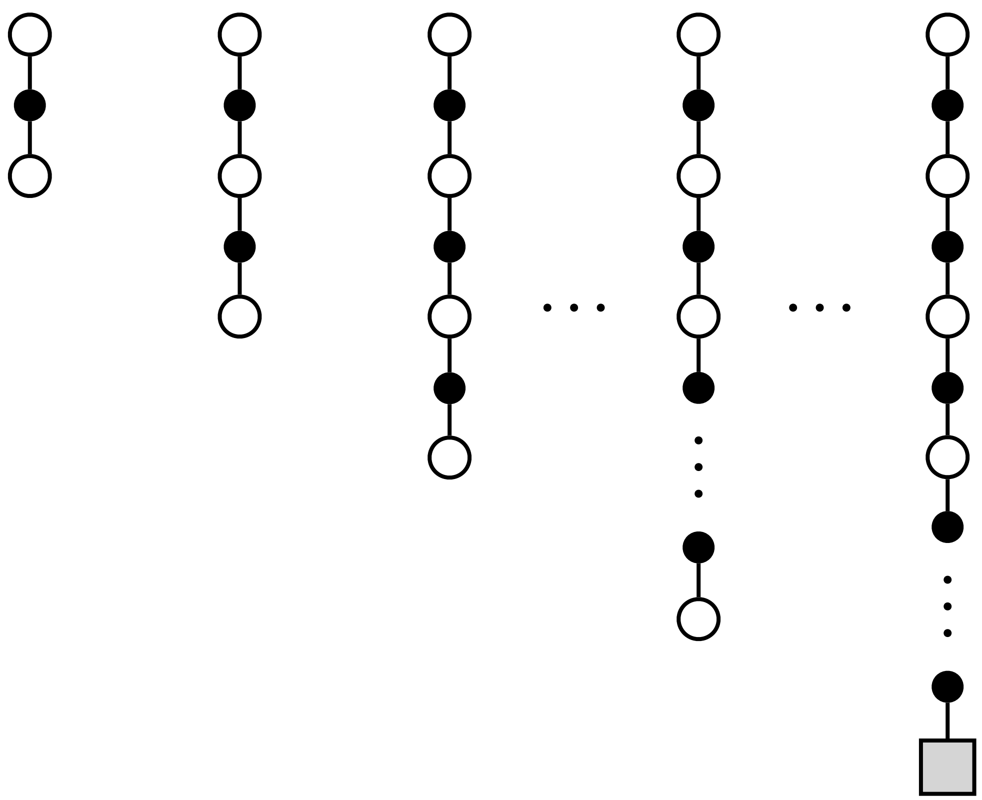


Outline

- Introduction
- Monte-Carlo Learning
- Temporal-Difference Learning
- TD(λ)

n-Step Prediction and Return

n-step Prediction: Let TD target look **n steps** into the future



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

TD(0)	$n = 1$	$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$
	$n = 2$	$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$
		$\vdots$
MC	$n = \infty$	$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$

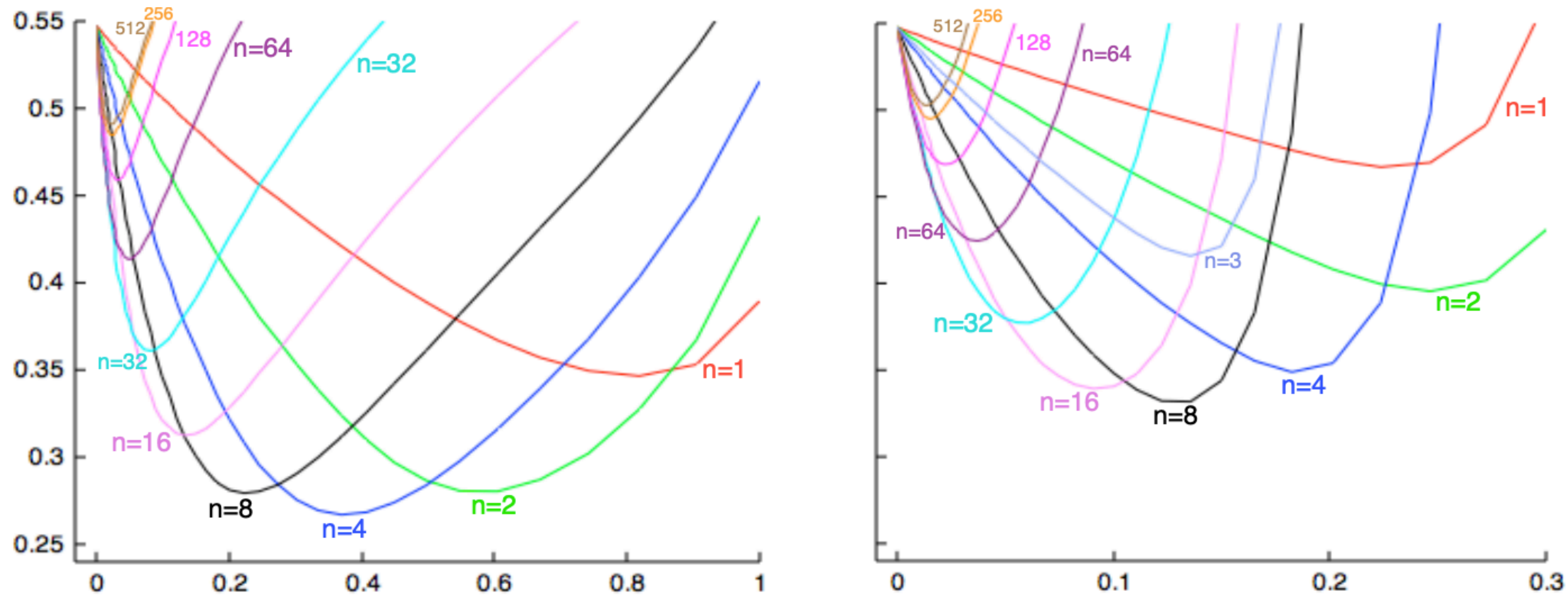
Define the n-step return $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$

n-step temporal-difference learning $V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$

Example - Large Random Walk

Online n-step TD   Offline n-step TD

RMS error, averaged over first 10 episodes



Step size α Step size α

Observations

- **Online methods** generally worked best on this task, reaching lower levels of absolute error
- Methods with an **intermediate value of n** worked best
    - Generalization of TD and Monte Carlo methods to **n-step methods** can potentially perform better than either of the two extreme methods

Averaging n-Step Returns

Motivation

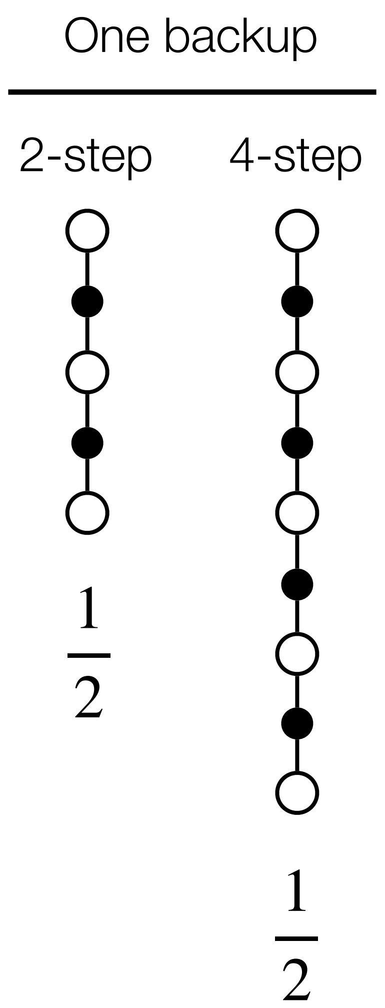
- Methods with an **intermediate value of n** worked best
  - How to pick **the best n**?

Averaging n-step returns

- We can **average n-step returns** over different n
  - e.g., average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

  - Combines information from two different time-steps

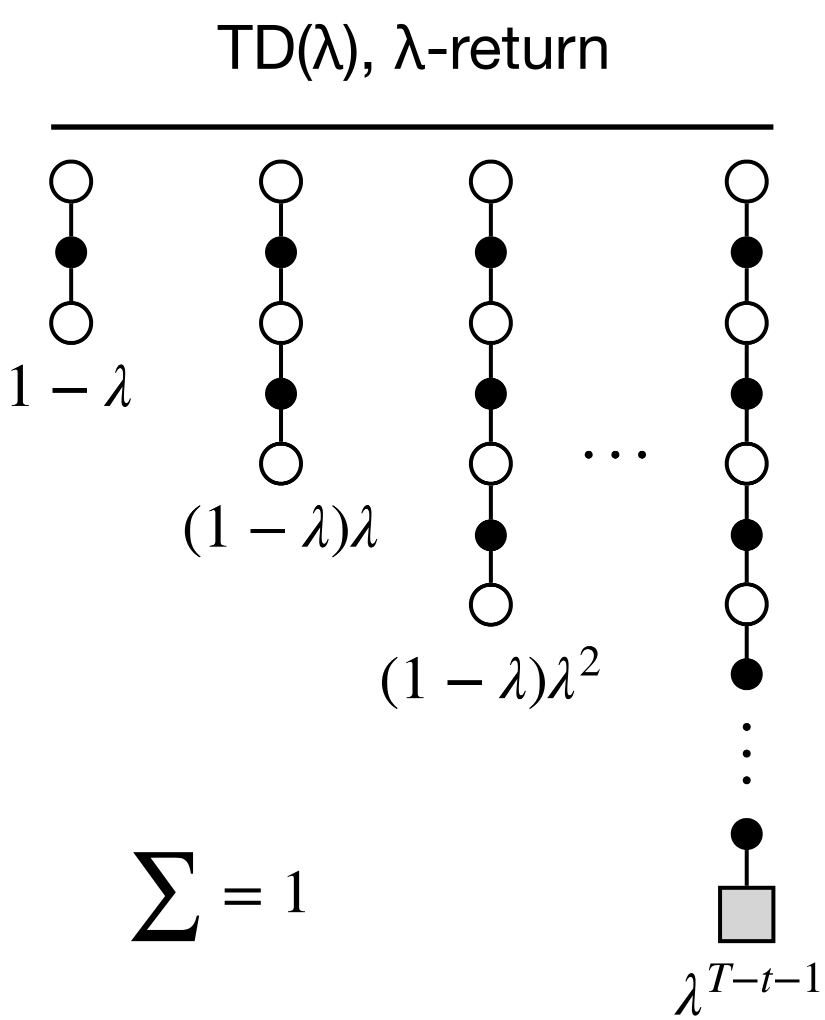


Can we efficiently combine information from **all time-steps**?

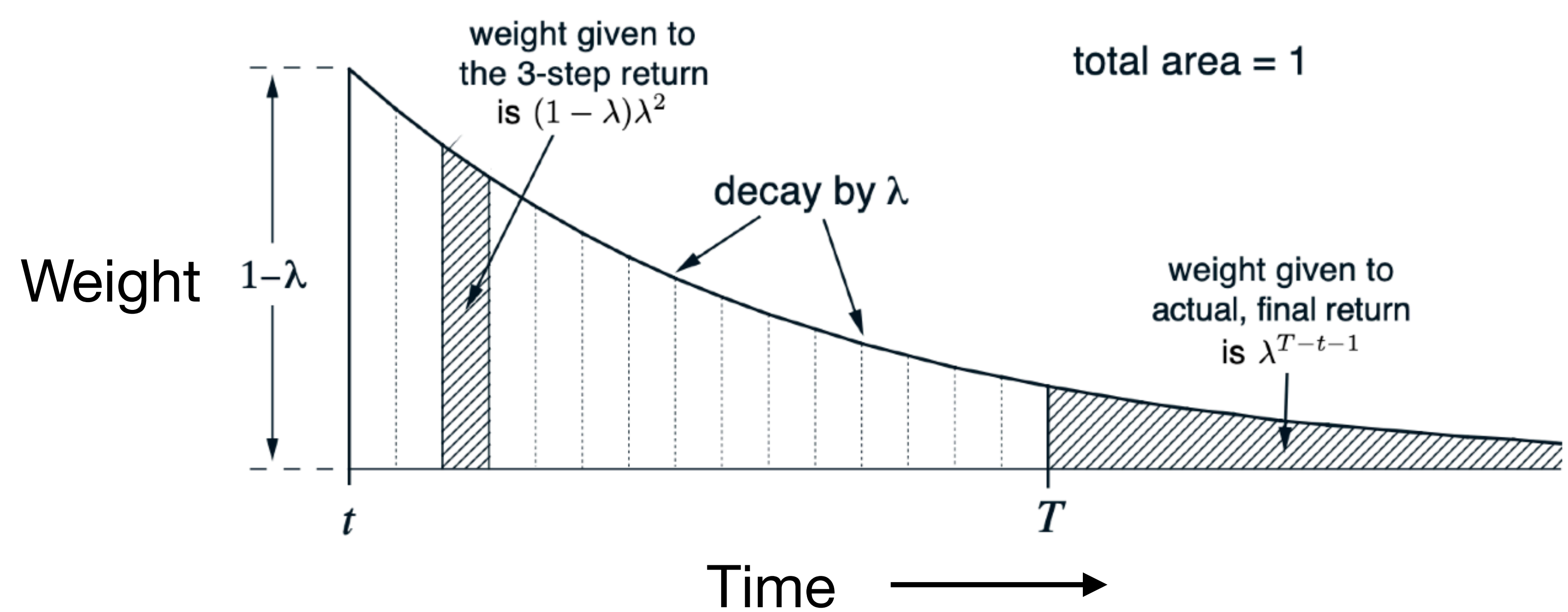
λ-return

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

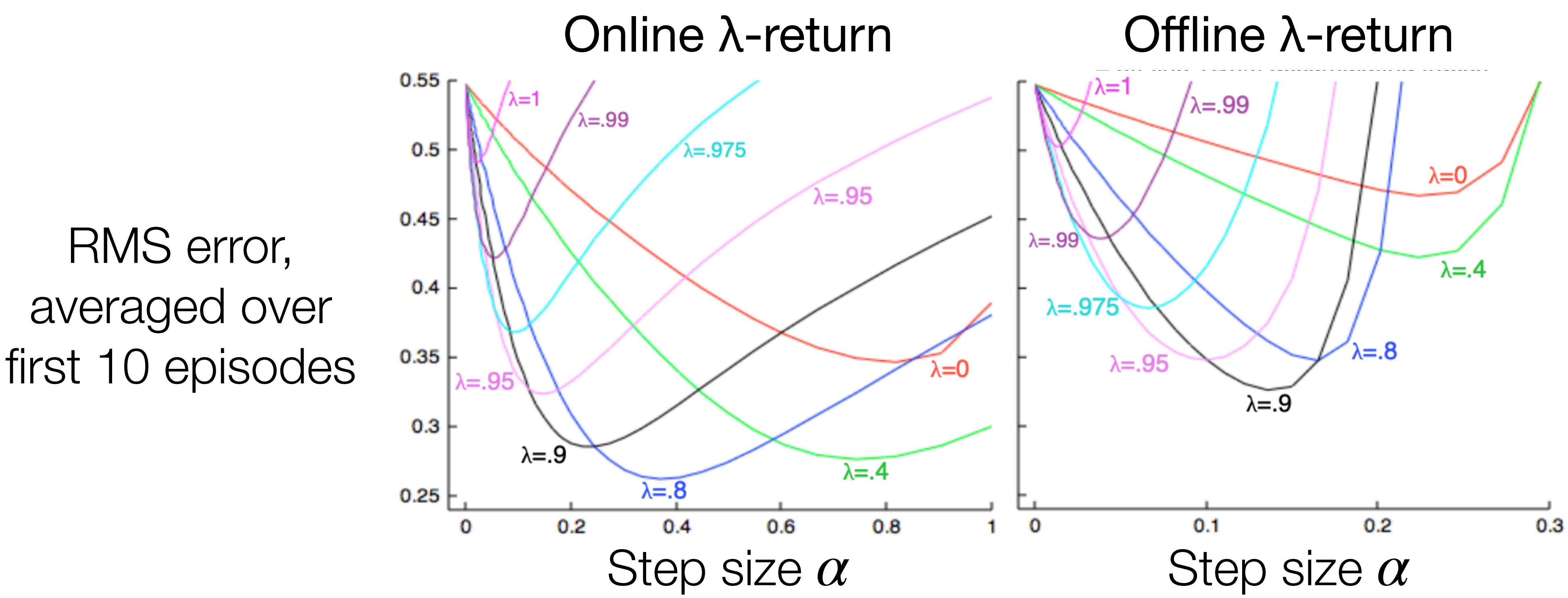
- Using weight $(1-\lambda)\lambda^{n-1}$: $G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$
- TD(λ) backups: $V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right)$

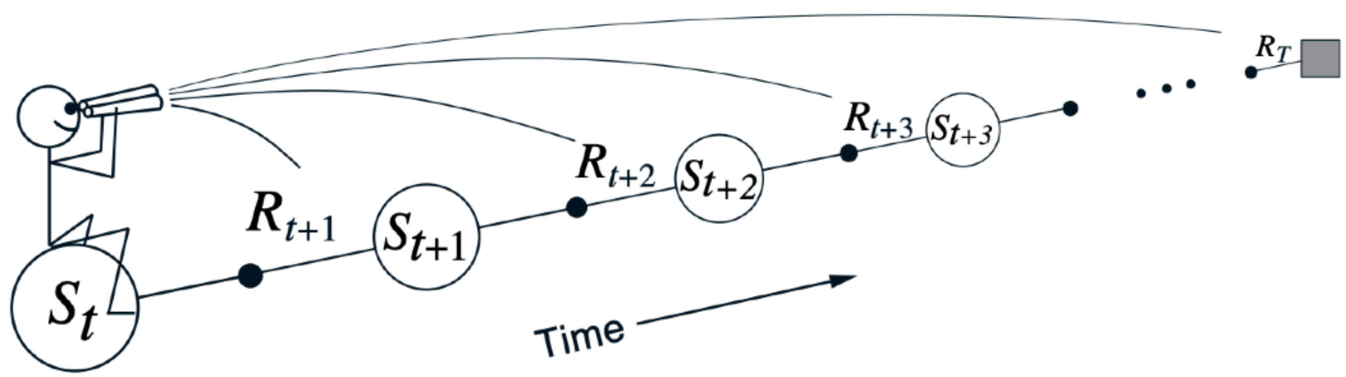


Weighting given in the λ-return to each of the n-step returns



Forward-view TD(λ)



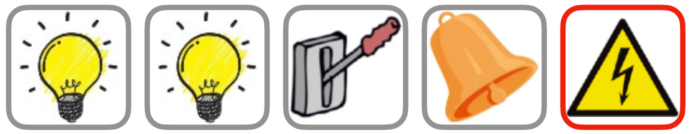


- Update value function towards the λ-return
- Forward-view looks into the future to compute G_t^{λ}
- Like MC, can only be computed from complete episodes

Backward-view TD(λ) and Eligibility Traces

Goal: update online, every step, from incomplete sequences

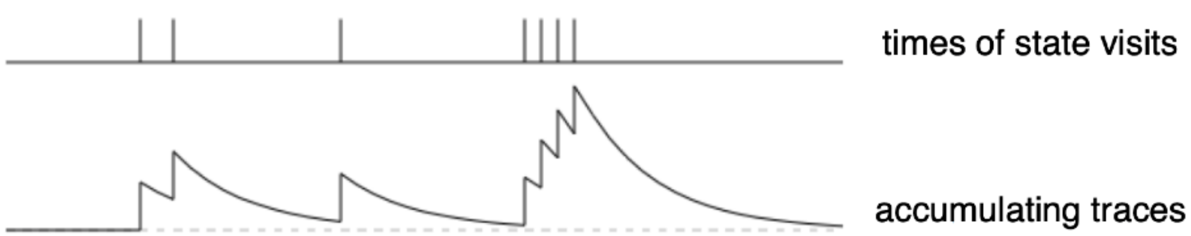
Eligibility Traces



- Credit assignment problem: did bell or light cause shock?
  - Frequency heuristic: assign credit to most frequent states
  - Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$E_0(s) = 0$$

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

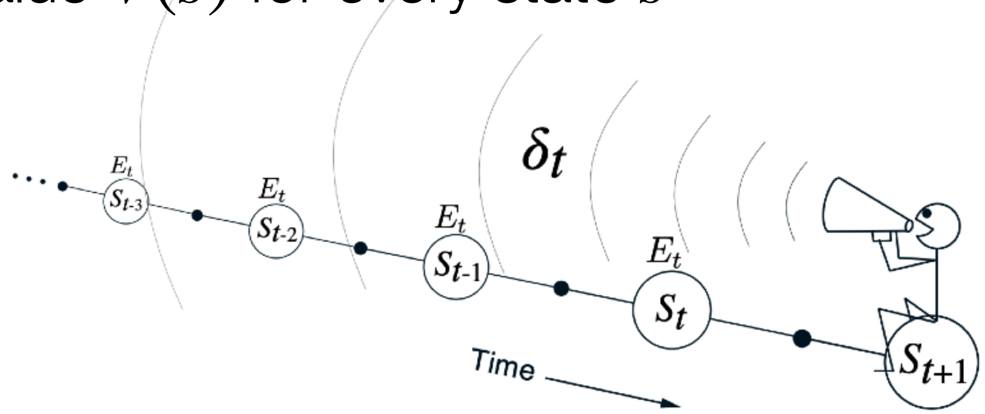


Backward-view TD(λ)

- Keep an **eligibility trace** for every state s and 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)

Theorem

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

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

λ=0

- Only current state is updated $E_t(s) = \mathbf{1}(S_t = s)$ and $V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$
- This is exactly equivalent to TD(0) update $V(s) \leftarrow V(s) + \alpha \delta_t$

MC and TD(1)

λ=1

- Credit is deferred until end of episode
- Consider episodic environments with offline updates
- Over the course of an episode, total update for TD(1) is the same as total update for MC

Consider an episode where s is visited once at time-step k,

- TD(1) eligibility trace discounts time since visit,

$$E_t(s) = \gamma E_{t-1}(s) + \mathbf{1}(S_t = s) = \begin{cases} 0, & \text{if } t < k \\ \gamma^{t-k}, & \text{if } t \geq k \end{cases}$$

TD(1) updates accumulate error online

$$\sum_{t=1}^{T-1} \alpha \delta_t E_t(s) = \alpha \sum_{t=k}^{T-1} \gamma^{t-k} \delta_t = \alpha(G_k - V(S_k))$$

By the end of episode it accumulates total error

$$\delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \ldots + \gamma^{T-1-k} \delta_{T-1}$$

MC and TD(1)

When λ = 1, sum of TD errors telescopes into MC error,

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

λ=1

- TD(1) is roughly equivalent to every-visit Monte-Carlo
  - Except that error is accumulated online, step-by-step
- If value function is only updated offline at end of episode
  - Then total update is exactly the same as MC

Equivalence of Forward and Backward TD(λ)

Consider an episode where s is visited once at time-step k,

- TD(λ) eligibility trace discounts time since visit,

$$E_t(s) = \gamma E_{t-1}(s) + \mathbf{1}(S_t = s) = \begin{cases} 0, & \text{if } t < k \\ (\gamma\lambda)^{t-k}, & \text{if } t \geq k \end{cases}$$

Backward TD(λ) updates accumulate error online

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \alpha \sum_{t=k}^{T} (\gamma\lambda)^{t-k} \delta_t = \alpha(G_k^\lambda - V(S_k))$$

By end of episode it accumulates total error for λ-return

For multiple visits to s, $E(s)$ accumulates many errors

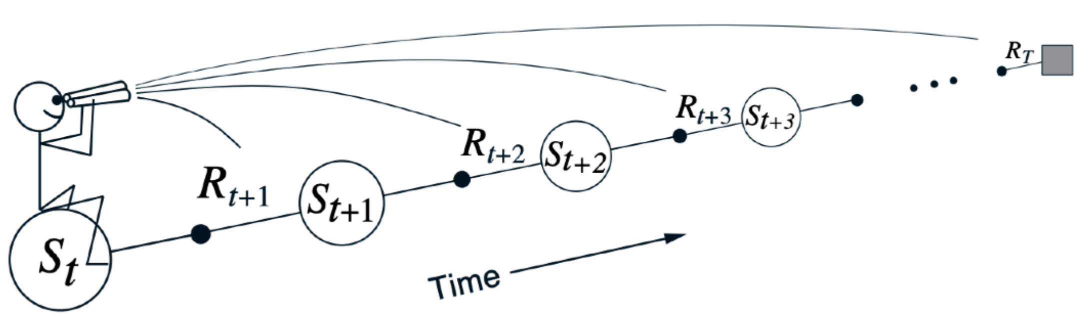
Equivalence of Forward and Backward TD(λ)

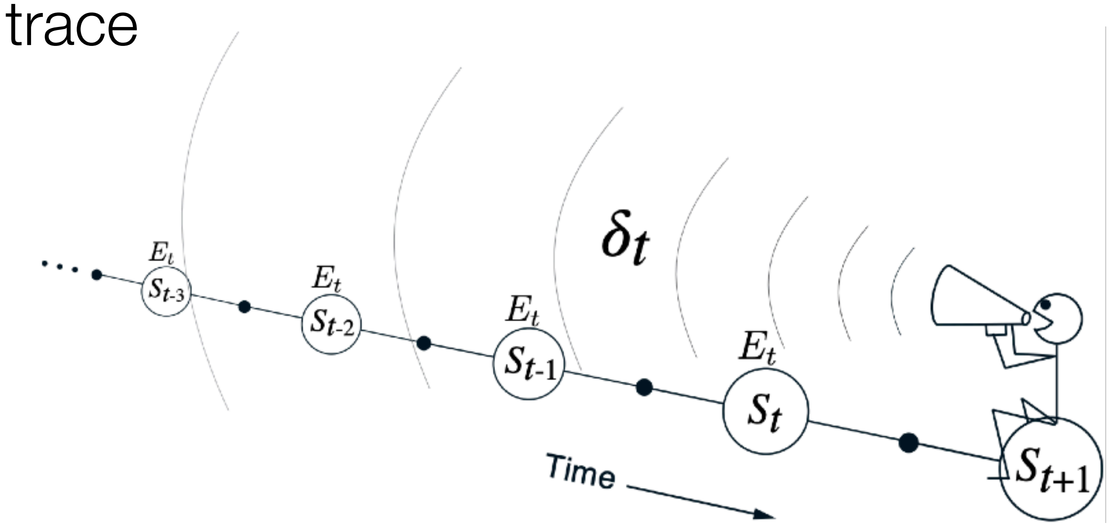
Offline updates

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

Online updates

- TD(λ) updates are applied online at each step within episode
- Forward and backward-view TD(λ) are slightly different
- Seijen, Harm, and Rich Sutton. "[True online TD (lambda).]()" *ICML*, 2014.
  - Exact online TD(λ) achieves perfect equivalence
  - By using a slightly different form of eligibility trace





Summary of Forward and Backward TD(λ)

Offline updates	$\lambda=0$	$\lambda \in [0, 1]$)	$\lambda=1$
Backward view **Forward view**	TD(0) = TD(0)	TD(λ) = Forward TD(λ)	TD(1) = MC
Online updates	$\lambda=0$	$\lambda \in [0, 1]$)	$\lambda=1$
Backward view **Forward view** **Exact Online**	TD(0) = TD(0) = TD(0)	TD(λ) ≠ Forward TD(λ) = Exact Online TD(λ)	TD(1) ≠ MC = Exact Online TD(1)

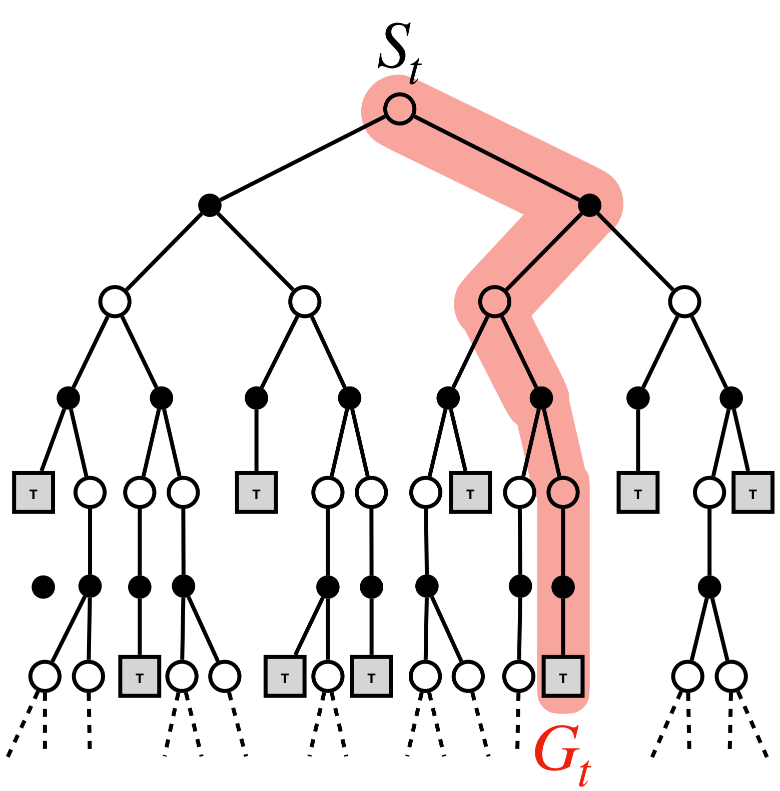
= here indicates equivalence in total update at end of episode

Summary - Model-Free Prediction

Monte-Carlo backup

- Based on **the entire sequence** of observed rewards until the end of the episode

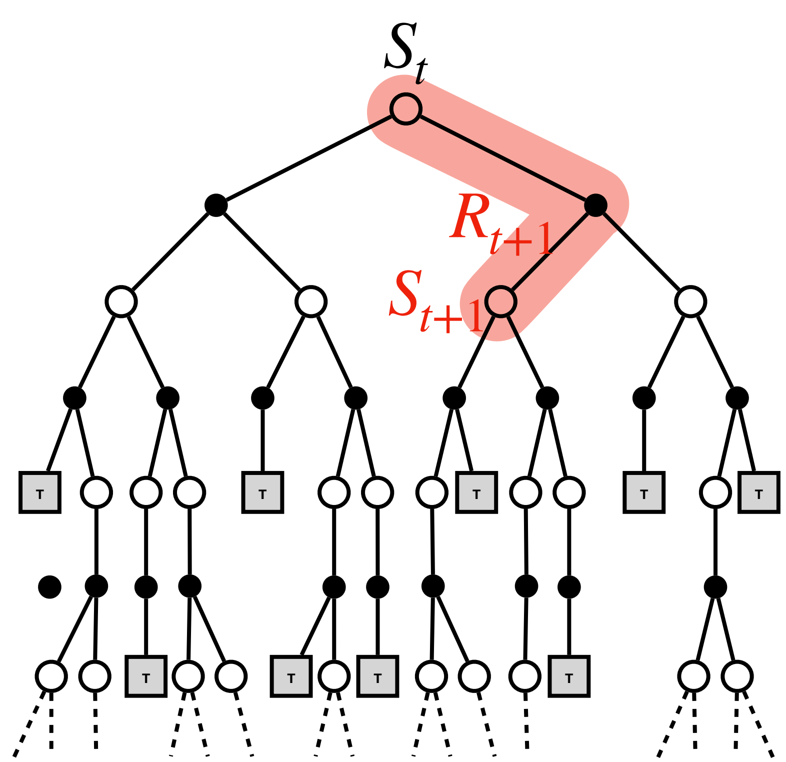
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



Temporal-Difference Backup

- Based on just the one **next reward**, using **the value of the state one step later** as a proxy for the remaining rewards

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



TD(λ) Backup

- Based on **an intermediate number of rewards**: more than one, but less than all of them until termination

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

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