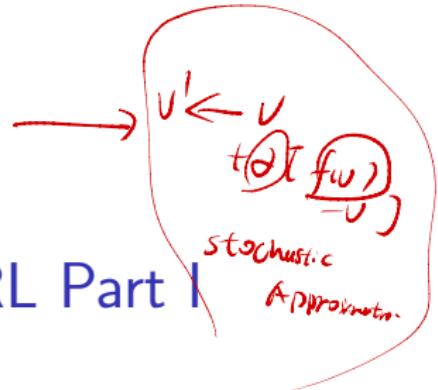


$$V = E[f(V)]$$

$$V' \leftarrow V + \alpha [E[f(V)] - V]$$



## Lecture 7: Model-Free RL Part I

Ziyu Shao

School of Information Science and Technology  
ShanghaiTech University

April 23, 2025

# Outline

- 1 Introduction
- 2 Monte-Carlo Learning
- 3 Temporal-Difference Learning
- 4 n-step TD Methods
- 5  $\text{TD}(\lambda)$
- 6 References

# Outline

1 Introduction

2 Monte-Carlo Learning

3 Temporal-Difference Learning

4 n-step TD Methods

5  $\text{TD}(\lambda)$

6 References

# Model-Free Reinforcement Learning

- Last lecture:
  - ▶ Planning by dynamic programming
  - ▶ Solve a known MDP
- This lecture:
  - ▶ Model-free prediction
  - ▶ Estimate the value function of an unknown MDP
- Next lecture:
  - ▶ Model-free control
  - ▶ Optimize the value function of an unknown MDP

# Outline

1 Introduction

2 Monte-Carlo Learning

3 Temporal-Difference Learning

4 n-step TD Methods

5  $\text{TD}(\lambda)$

6 References

# Monte-Carlo Reinforcement Learning



Finite stages  
episode

- 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
- To learn values & policies, MC can be used in two ways:
  - ▶ *model-free*: no model necessary and still attains optimality
  - ▶ simulated: needs only a simulation, not a full model
- Caveat: can only apply MC to *episodic* MDPs
  - ▶ All episodes must terminate

# Monte-Carlo Policy Evaluation

- Goal: learn  $v_\pi$  from episodes of experience under policy  $\pi$

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

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

# Monte-Carlo Policy Evaluation

- Goal: learn  $v_\pi(s)$
- Given: some number of episodes under  $\pi$  which contains  $s$
- Idea: average returns observed after visits to  $s$
- Every-Visit MC: average returns for every time  $s$  is visited in an episode
- First-visit MC: average returns only for first time  $s$  is visited in an episode
- Both converge asymptotically

# 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  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = \frac{S(s)}{N(s)}$
- By law of large numbers,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$

# First-Visit Monte-Carlo Policy Evaluation

First-visit MC prediction, for estimating  $V \approx v_\pi$

Input: a policy  $\pi$  to be evaluated

Initialize:

$V(s) \in \mathbb{R}$ , arbitrarily, for all  $s \in S$

$Returns(s) \leftarrow$  an empty list, for all  $s \in S$

Loop forever (for each episode):

Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :

$G \leftarrow \gamma G + R_{t+1}$

Unless  $S_t$  appears in  $S_0, S_1, \dots, S_{t-1}$ :

Append  $G$  to  $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

# 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  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- Again,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$

# Incremental Mean

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

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\&= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\&= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\&= \boxed{\mu_{k-1}} + \boxed{\frac{1}{k} (x_k - \mu_{k-1})}\end{aligned}$$

# Incremental Monte-Carlo Updates

- Update  $V(s)$  incrementally after episode  $S_1, A_1, R_2, \dots, S_T$
- For each state  $S_t$  with return  $G_t$

$$N(S_t) \leftarrow \underline{N(S_t) + 1}$$
$$V(S_t) \leftarrow \underline{V(S_t) + \frac{1}{N(S_t)}(\boxed{G_t} - V(S_t))}$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha(\boxed{G_t} - V(S_t))$$

$\alpha_t$

# Monte-Carlo Estimation of Action Values

- Monte Carlo is most useful when a model is not available: we want to learn  $q_*$
- $\underline{q_\pi(s, a)}$ : average return starting from state  $s$  and action  $a$  following policy  $\pi$
- Converges asymptotically if every state-action pair is visited
- Exploring Starts: every state-action pair has a non-zero probability of being the starting pair

# Backup Diagram for Monte-Carlo

- Entire rest of episode included
- Only one choice considered at each state (unlike DP)
- thus, there will be an explore/exploit dilemma
- Does not bootstrap from successor states's values (unlike DP)
- Time required to estimate one state does not depend on the total number of states



# Outline

- 1 Introduction
- 2 Monte-Carlo Learning
- 3 Temporal-Difference Learning
- 4 n-step TD Methods
- 5  $\text{TD}(\lambda)$
- 6 References

# Temporal-Difference Learning

Bootstrapping in life : to lift himself up by his bootstraps.

Bootstrapping in statistics : (97) Bradley Efron.

- 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

Resampling

Exist samples : 2, 4, 5, 6, 6,

Resample with replacement:

2, 5, 5, 6, 6  
4, 5, 2, 2, 5

# MC and TD

- Goal: learn  $v_\pi$  online from experience under policy  $\pi$
- Incremental every-visit Monte-Carlo

Update value  $V(S_t)$  toward *actual return*  $G_t$

$$V_\pi(s) = E_\pi[G_t | S_t=s]$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

error

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value  $V(S_t)$  toward *estimated return*  $R_{t+1} + \gamma V(S_{t+1})$

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t=s]$$
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

►  $R_{t+1} + \gamma V(S_{t+1})$  is called the *TD target*

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

# Advantages and Disadvantages of MC vs. TD

- TD can learn before knowing the final outcome
  - ▶ TD can learn online after every step (less memory & peak computation)
  - ▶ 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

# Bias/Variance Trade-Off

- Return  $\underline{G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T}$  is *unbiased* estimate of  $v_\pi(\underline{S_t})$
- True TD target  $R_{t+1} + \gamma \cancel{v_\pi(S_{t+1})}$  is *unbiased* estimate of  $v_\pi(S_t)$
- TD target  $\underline{R_{t+1} + \gamma V(S_{t+1})}$  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

# Advantages and Disadvantages of MC vs. TD (2)

- MC has high variance, zero bias
  - ▶ 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
  - ▶ Usually more efficient than MC
  - ▶ TD(0) converges to  $v_\pi(s)$
  - ▶ (but not always with function approximation)
  - ▶ More sensitive to initial value

# Advantages and Disadvantages of MC vs. TD (3)

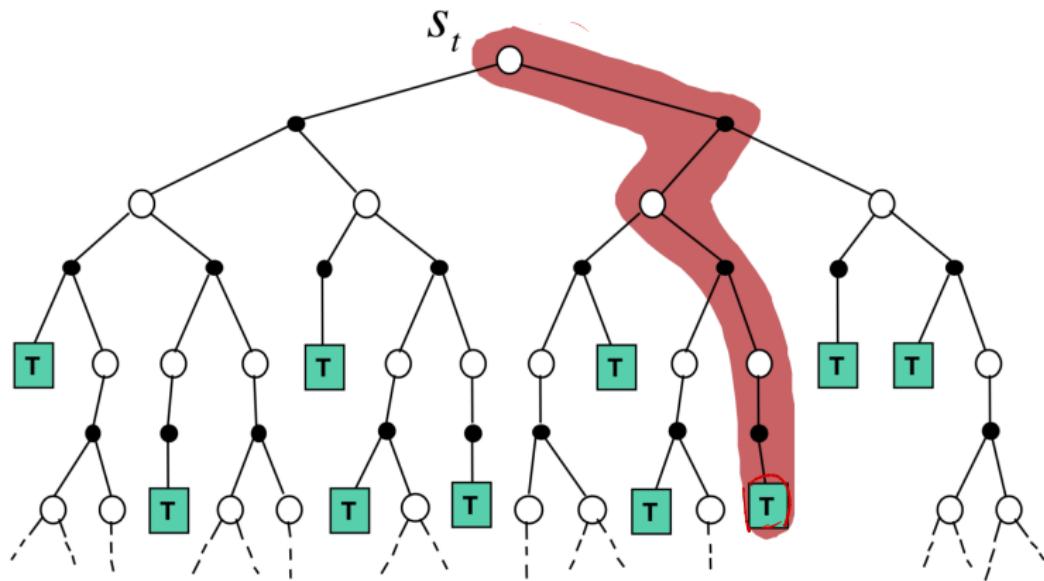
- TD exploits Markov property
  - ▶ Usually more efficient in Markov environments
- MC does not exploit Markov property
  - ▶ Usually more effective in non-Markov environments
- MC has lower error on past data, but higher error on future data

# Bellman Backup

- The term “Bellman backup” comes up quite frequently in the RL literature.
- The Bellman backup for a state (or a state-action pair) is the right-hand side of the Bellman equation:  
the reward-plus-next-value.
- Under different algorithms, we obtain
  - ▶ Monte-Carlo Backup
  - ▶ Temporal-Difference Backup
  - ▶ Dynamic Programming Backup

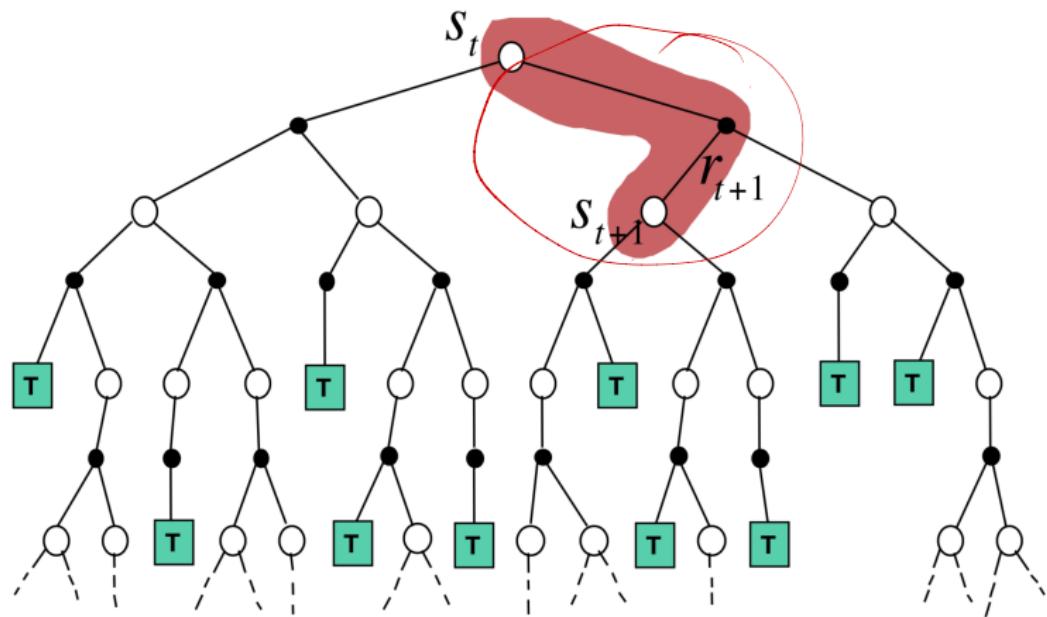
# Monte-Carlo Backup

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



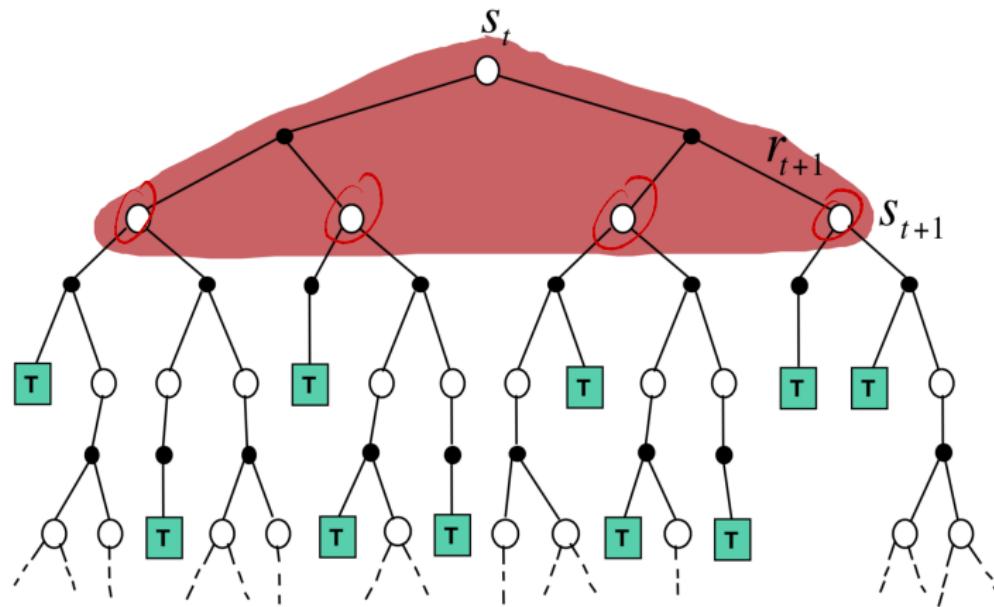
# Temporal-Difference Backup

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



# Dynamic Programming Backup

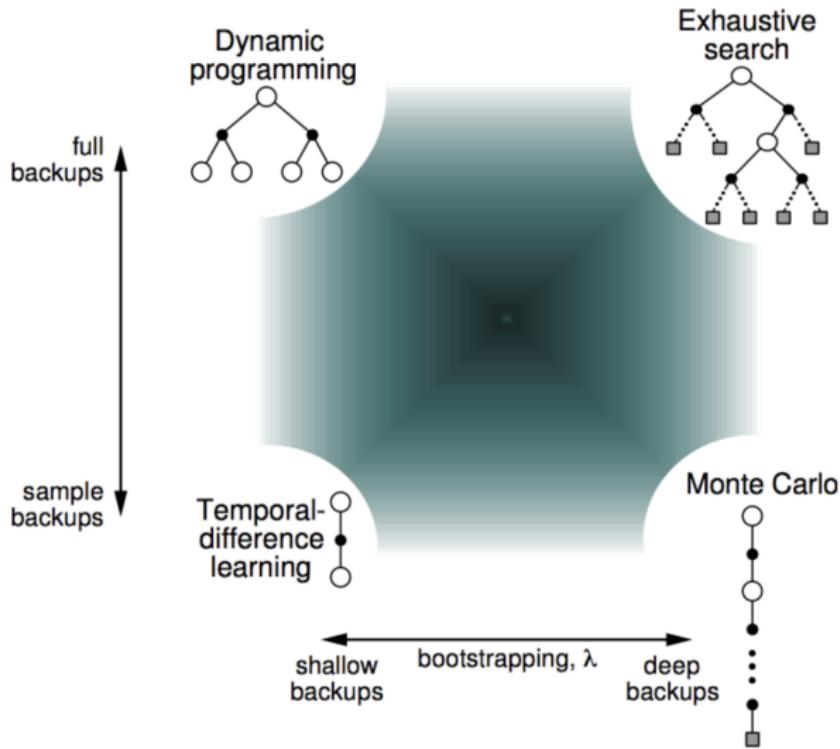
$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$



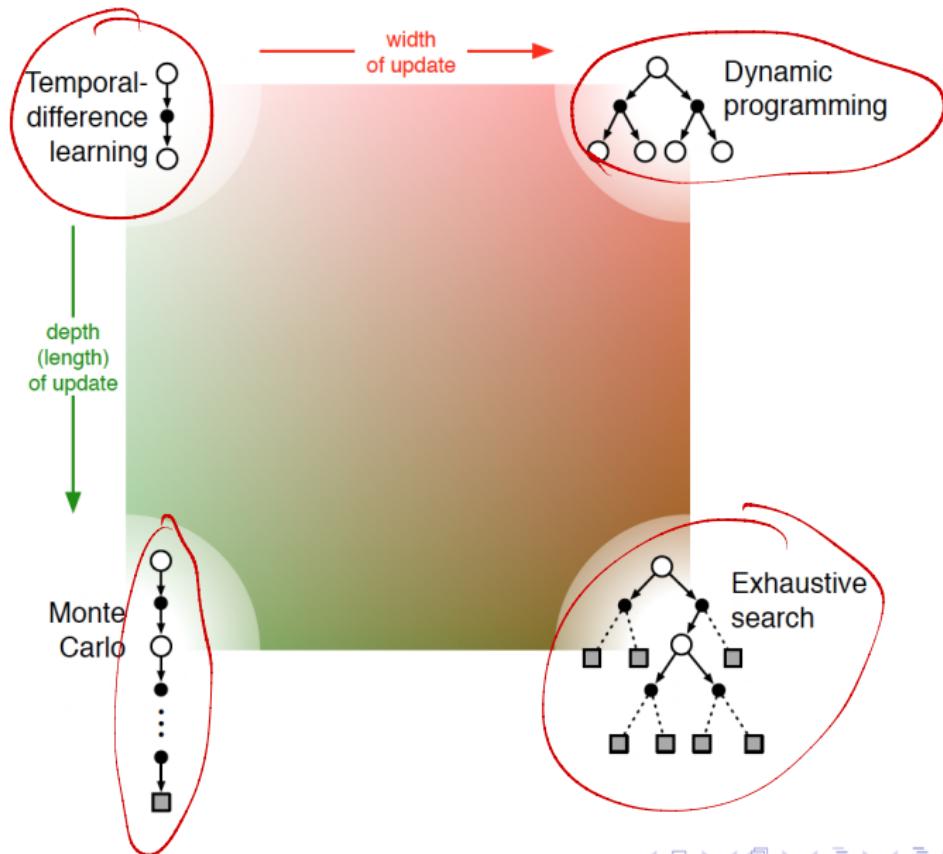
# Bootstrapping and Sampling

- Bootstrapping: update involves an estimate
  - ▶ MC does not bootstrap
  - ▶ DP bootstraps
  - ▶ TD bootstraps
- Sampling: update samples an expectation
  - ▶ MC samples (episode)
  - ▶ DP does not sample
  - ▶ TD samples (stage)

# Unified View of Reinforcement Learning



# Unified View of Reinforcement Learning

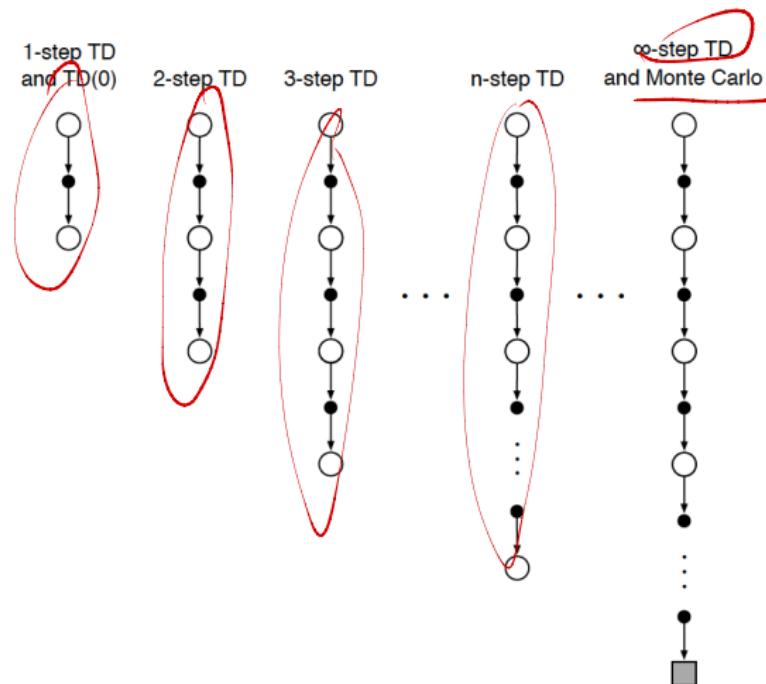


# Outline

- 1 Introduction
- 2 Monte-Carlo Learning
- 3 Temporal-Difference Learning
- 4 n-step TD Methods
- 5  $\text{TD}(\lambda)$
- 6 References

# $n$ -Step Prediction

- Let TD target look  $n$  steps into the future



# *n*-Step Return

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

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

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

⋮                ⋮

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

- Define the *n*-step return

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

- n*-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha \circled{(G_t^{(n)} - V(S_t))}$$

## *n*-step TD

- Recall the *n*-step return:

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \quad n \geq 1, 0 \leq t < T-n$$

- Of course, this is not available until time  $t+n$
- The natural algorithm is thus to **wait** until then

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

- This is called ***n*-step TD**

# n-step TD Algorithm

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

Initialize  $V(s)$  arbitrarily,  $s \in \mathcal{S}$

Parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$

All store and access operations (for  $S_t$  and  $R_t$ ) can take their index mod  $n$

Repeat (for each episode):

    Initialize and store  $S_0 \neq$  terminal

$T \leftarrow \infty$

    For  $t = 0, 1, 2, \dots :$

        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  $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$     ( $\tau$  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:  $G \leftarrow G + \gamma^n V(S_{\tau+n})$

$(G_\tau^{(n)})$

$$V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$$

    Until  $\tau = T - 1$

# Error Reduction Property

- Error reduction property of n-step returns

$$\underbrace{\max_s \left| \mathbb{E}_\pi \left[ G_t^{(n)} \middle| S_t = s \right] - v_\pi(s) \right|}_{\text{Maximum error using } n\text{-step return}} \leq \underbrace{\gamma^n \max_s |V_t(s) - v_\pi(s)|}_{\text{Maximum error using V}}$$

- Using this property, we can show that n-step TD methods converge
- n-step TD methods: a family of sound methods including one-step TD methods & MC methods as extreme members

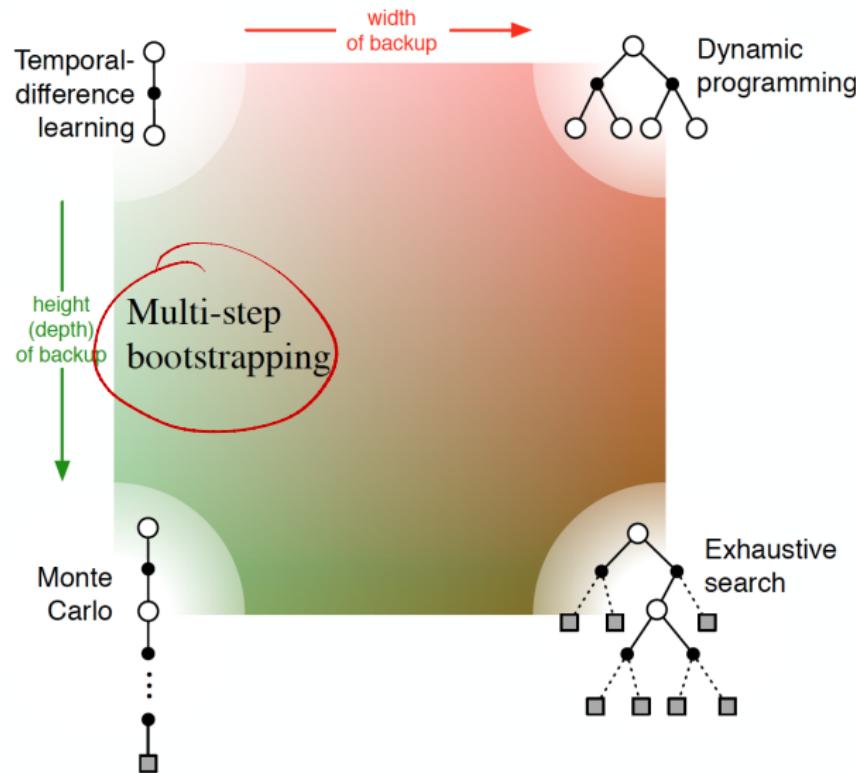
# Summary of n-step TD Methods

- Generalize Temporal-Difference and Monte Carlo learning methods, sliding from one to the other as  $n$  increases
  - ▶  $n = 1$  is TD
  - ▶  $n = \infty$  is MC
  - ▶ an intermediate  $n$  is often much better than either extreme
  - ▶ applicable to both continuing and episodic problems
- There is some cost in computation
  - ▶ need to remember the last  $n$  states
  - ▶ learning is delayed by  $n$  steps
  - ▶ per-step computation is small and uniform, like TD
- Everything generalizes nicely: error-reduction theory

# Outline

- 1 Introduction
- 2 Monte-Carlo Learning
- 3 Temporal-Difference Learning
- 4 n-step TD Methods
- 5 TD( $\lambda$ )
- 6 References

# Unified View

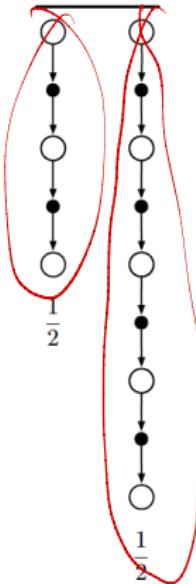


# Averaging $n$ -Step Returns

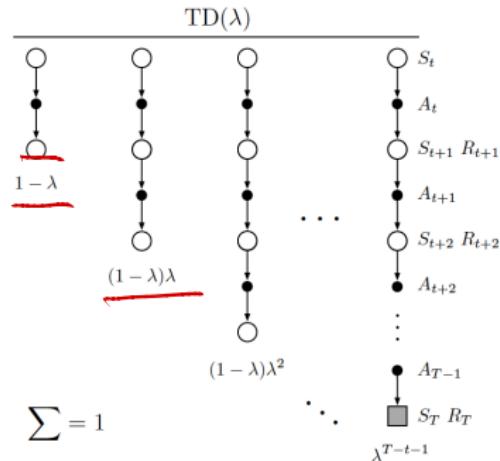
- We can average  $n$ -step returns over different  $n$
- e.g. average the 2-step and 4-step returns

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

- Combines information from two different time-steps
- Can we efficiently combine information from all time-steps?



## $\lambda$ -return



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

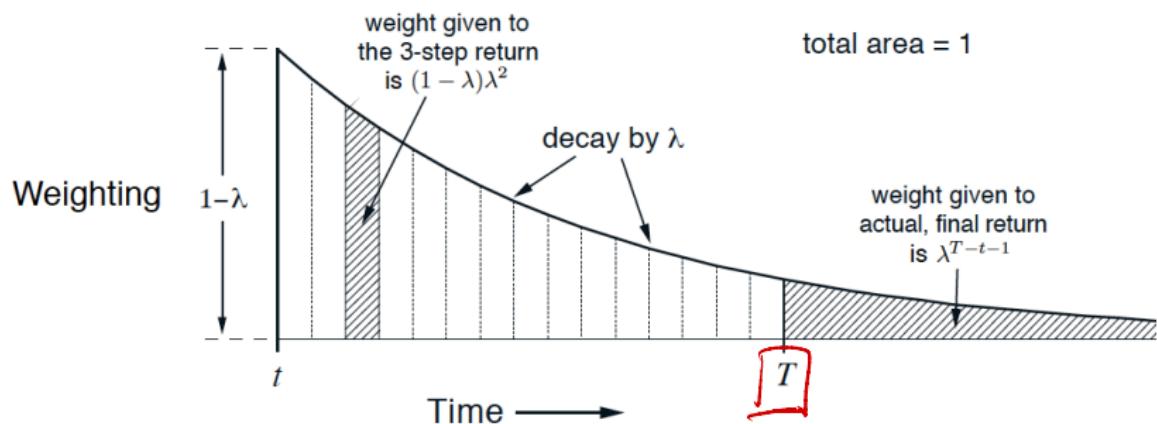
$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Forward-view TD( $\lambda$ )

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

$$\begin{aligned} \sum_{n=1}^{\infty} (1-\lambda) \cdot \lambda^{n-1} &= (1-\lambda) / \sum_{n=1}^{\infty} \lambda^n \\ &= (1-\lambda) / \frac{1}{1-\lambda} = 1 \end{aligned}$$

# TD( $\lambda$ ) Weighting Function



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

# Relation to TD(0) & MC

- The  $\lambda$ -return can be rewritten as:

$$G_t^\lambda = \underbrace{(1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)}}_{\text{Until termination}} + \underbrace{\lambda^{T-t-1} G_t}_{\text{After termination}}$$

- if  $\lambda = 1$ , you get the MC target:

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

- If  $\lambda = 0$ , you get the TD(0) target:

$$G_t^\lambda = (1 - 0) \sum_{n=1}^{T-t-1} 0^{n-1} G_t^{(n)} + 0^{T-t-1} G_t = G_t^{(1)}$$

# Summary of TD( $\lambda$ ) algorithms

- Another way of interpolating between MC and TD methods
- A way of implementing compound  $\lambda$ -return targets

# Outline

1 Introduction

2 Monte-Carlo Learning

3 Temporal-Difference Learning

4 n-step TD Methods

5  $\text{TD}(\lambda)$

6 References

# Main References

- Reinforcement Learning: An Introduction (second edition), R. Sutton & A. Barto, 2018.
- RL course slides from Richard Sutton, University of Alberta.
- RL course slides from David Silver, University College London.