

Lecture 10: Model-Free Prediction

Ziyu Shao

School of Information Science and Technology
ShanghaiTech University

May 11 & 13, 2020

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

$p_{s,s'}$
 r_s^a

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

- 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_π from episodes of experience under policy π

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

- 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 π 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 $V(s) = S(s)/N(s)$
- 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 π 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 π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

back-ward.

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 $N(s) \rightarrow \infty$

Blackjack Example

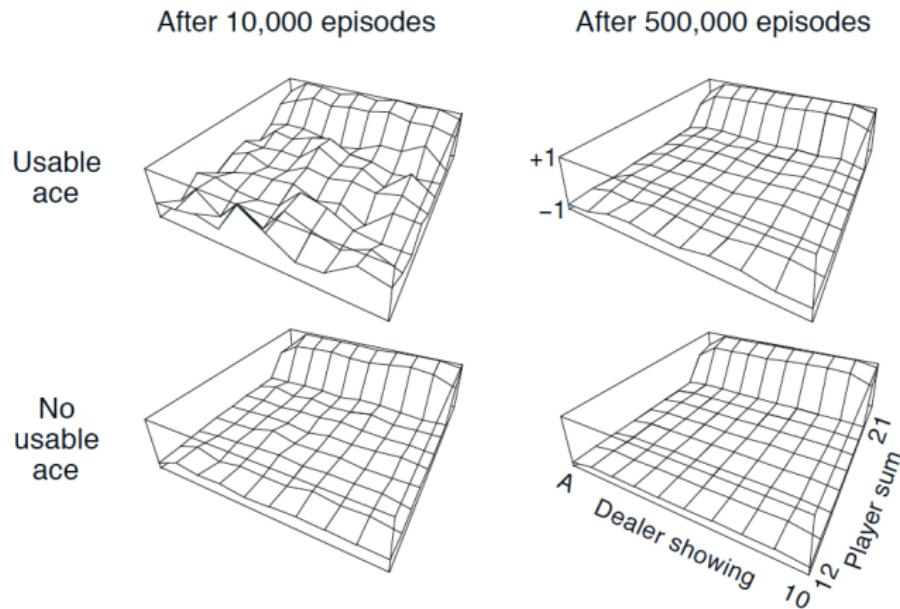


- States (200 of them):
 - ▶ Current sum (12-21)
 - ▶ Dealer's showing card (ace-10)
 - ▶ Do I have a “useable” ace? (yes-no)

Blackjack Example

- Action **stick**: Stop receiving cards (and terminate)
- Action **twist**: Take another card (no replacement)
- Reward for **stick**:
 - ▶ +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
- Reward for **twist**:
 - ▶ -1 if sum of cards > 21 (and terminate)
 - ▶ 0 otherwise
- Transitions: automatically **twist** if sum of cards < 12

Blackjack Value Function after Monte-Carlo Learning



Policy: **stick** if sum of cards ≥ 20 , otherwise **twist**

Incremental Mean

The mean μ_1, μ_2, \dots of a sequence x_1, x_2, \dots can be computed incrementally,

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j && \text{batch} \\ &= \frac{1}{k} \left(x_k + \underbrace{\sum_{j=1}^{k-1} x_j}_{\text{---}} \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \underbrace{\mu_{k-1}}_{\text{---}} + \frac{1}{k} (x_k - \mu_{k-1}) && \text{incremental.}\end{aligned}$$

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

$$N(S_t) \leftarrow \underbrace{N(S_t) + 1}_{\text{Count}}$$
$$V(S_t) \leftarrow \underbrace{V(S_t)}_{\text{Old estimate}} + \frac{1}{\underbrace{N(S_t)}_{\text{Count}}} (G_t - V(S_t))$$

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

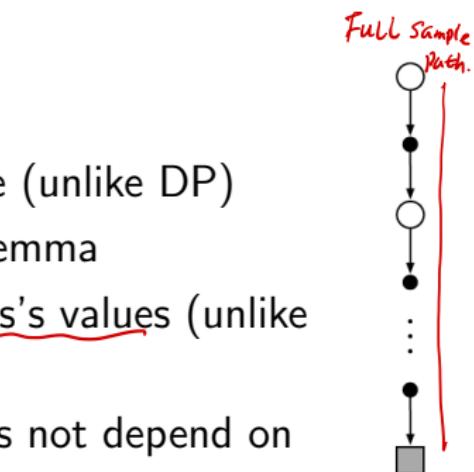
$$V(S_t) \leftarrow V(S_t) + \underbrace{\alpha}_{\text{step-size}} (G_t - V(S_t))$$

Monte-Carlo Estimation of Action Values

- Monte Carlo is most useful when a model is not available: we want to learn q_*
- $q_\pi(s, a)$: average return starting from state s and action a following policy π
- 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

1979 · Bradley Efron (resampling)
Existing samples: 2, 4, 5, 6, 6

- 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 with replacement

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

bootstrap
samples.

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo $v_{\pi}(s) = \mathbb{E}[G_t | S_{t=s}]$
 - ▶ Update value $V(S_t)$ toward *actual return* G_t
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$
- Simplest temporal-difference learning algorithm: $\text{TD}(0)$
 - ▶ 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*

Driving Home Example

MC update:

Predicted
Time to Go
43

38 :
23 :
13 :
3 :
0 :

Predicted
Total Time
43
43
43
43
43

<u>State</u>	<u>Elapsed Time</u> (minutes)	<u>Predicted</u> <u>Time to Go</u>	<u>Predicted</u> <u>Total Time</u>
leaving office, <u>friday at 6</u>	0	<u>30</u>	<u>30</u>
reach car, <u>raining</u>	5	<u>35</u>	<u>40</u>
exiting <u>highway</u>	<u>20</u>	<u>15</u>	<u>35</u>
<u>2ndary road, behind truck</u>	<u>30</u>	<u>10</u>	<u>40</u>
entering <u>home street</u>	<u>40</u>	<u>3</u>	<u>43</u>
arrive home	<u>43</u>	<u>0</u>	<u>43</u>

TD update

Predicted
Time to Go

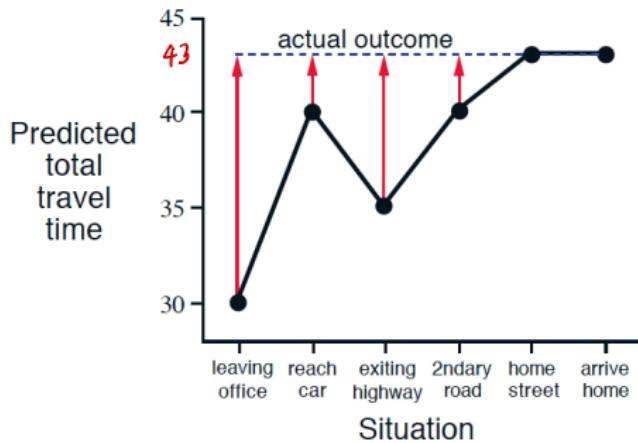
40
30
20
13
3
0

Predicted
Total Time

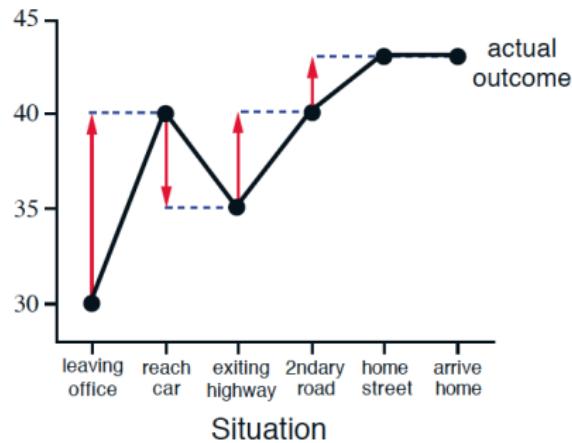
40
35
40
43
43
43

Driving Home Example: MC vs. TD

Changes recommended by MC methods ($\alpha = 1$)



Changes recommended by TD methods ($\alpha = 1$)



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

TD: Online update.

MC: batch update.

Bias/Variance Trade-Off

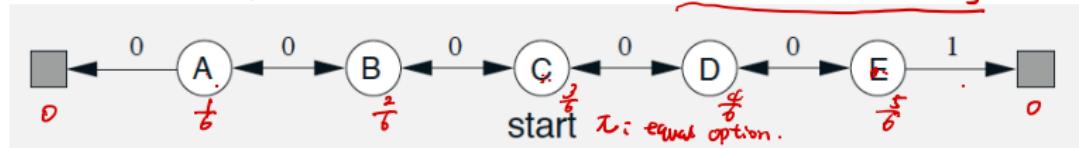
- Return $\underline{G_t} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(\underline{S_t})$
- True TD target $\underline{R_{t+1}} + \gamma v_\pi(\underline{S_{t+1}})$ is unbiased estimate of $v_\pi(\underline{S_t})$
- TD target $\underline{R_{t+1}} + \gamma V(\underline{S_{t+1}})$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

Random Walk Example: MC vs. TD

Bellman Expectation Equation with MRP. $V(S_t) = \mathbb{E}[R_t + \gamma V(S_{t+1})]$ $\gamma=1$, undiscouted.



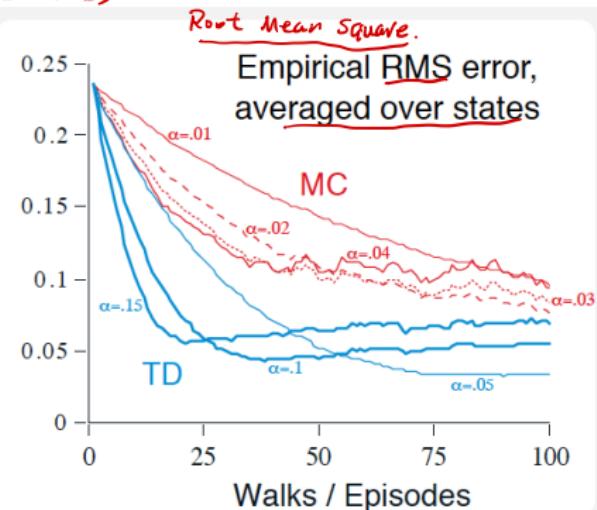
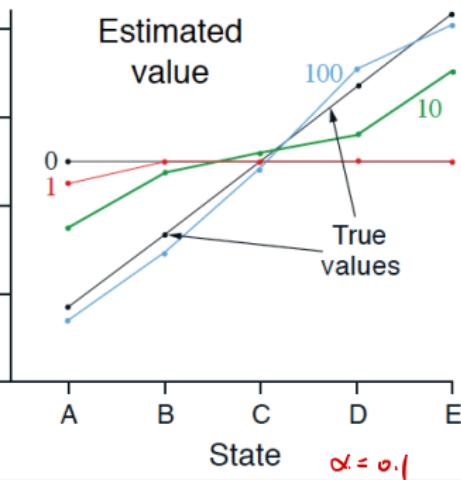
$$V_A = \frac{1}{2}(0+0) + \frac{1}{2}(0+V_B)$$

$$V_B = \frac{1}{2}(0+V_A) + \frac{1}{2}(0+V_C)$$

$$V_C = \frac{1}{2}(0+V_B) + \frac{1}{2}(0+V_D)$$

$$V_D = \frac{1}{2}(0+V_C) + \frac{1}{2}(0+V_E)$$

$$V_E = \frac{1}{2}(0+V_D) + \frac{1}{2}(1+0)$$



Batch MC and TD

- MC and TD converge: $V(s) \rightarrow v_\pi(s)$ as experience $\rightarrow \infty$
- But what about batch solution for finite experience?

$s_1^1, a_1^1, r_2^1, \dots, s_{T_1}^1$ *training set.*

\vdots

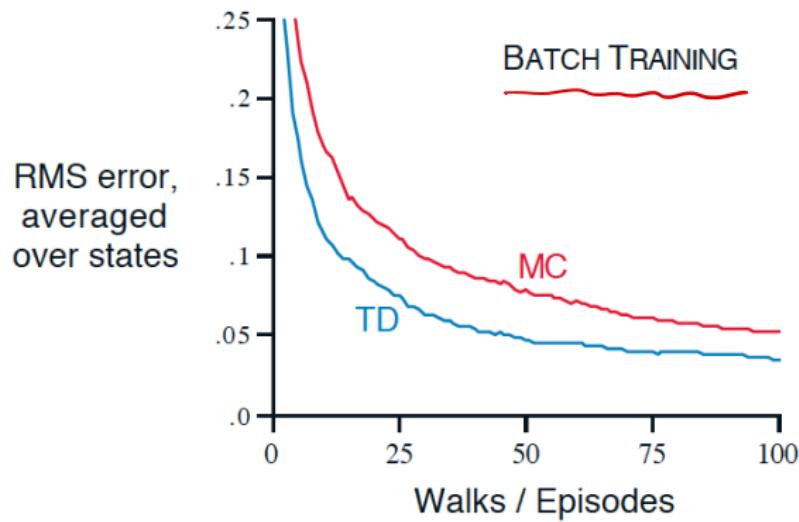
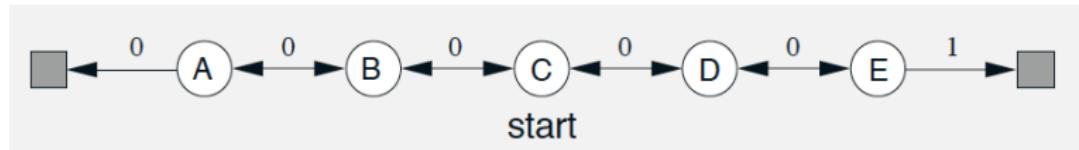
$s_1^K, a_1^K, r_2^K, \dots, s_{T_K}^K$

- ▶ e.g. Repeatedly sample episode $k \in [1, K]$
- ▶ Apply MC or TD(0) to episode k

Batch MC and TD

- For any finite Markov prediction task, under batch updating, TD converges for sufficiently small step-size parameter α
- Constant- α MC also converges under these conditions, but to a different answer!

Random Walk under Batch Updating



AB Example

Two states A, B ; no discounting; 8 episodes of experience

A, 0, B, 0

$$V(B) = \frac{3}{4} \cdot \left(\frac{6}{8} \right)$$

B, 1

B, 1

B, 1

$$V(A) = ?$$

B, 1

batch MC

$$V(A) = 0 \cdot \left(\frac{0}{1} \right)$$

B, 1

B, 1

B, 0

What is $V(A), V(B)$?

AB Example

batch TD(0)

Two states A, B ; no discounting; 8 episodes of experience

A, 0, B, 0

B, 1

B, 1

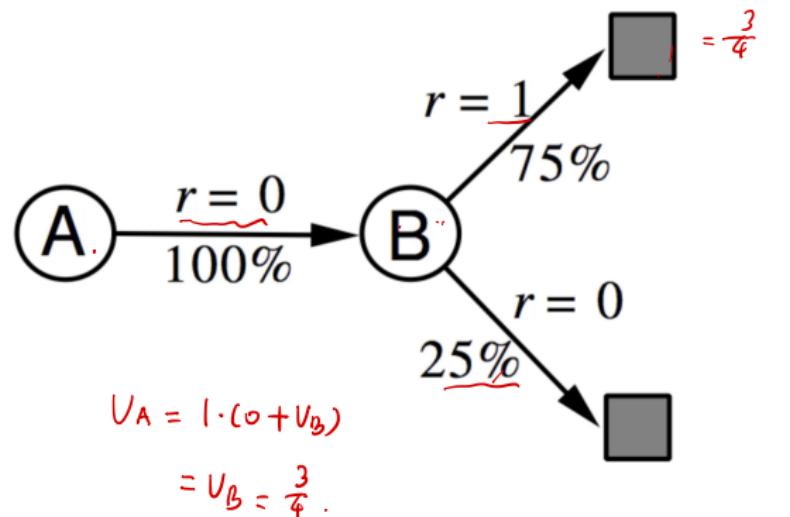
B, 1

B, 1

B, 1

B, 1

B, 0



What is $V(A), V(B)$?

AB Example

- The prediction that best matches the training data is $V(A) = 0$
 - ▶ This minimizes the mean-square-error on the training set zero-error, in training set.
 - ▶ This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set $V(A) = 0.75$
 - ▶ This is correct for the maximum likelihood estimate of a Markov model generating the data
 - ▶ i.e., if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts
 - ▶ This is called the certainty-equivalence estimate
 - ▶ This is what TD gets

Certainty Equivalence

- 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, $V(A) = 0$

- TD(0) converges to solution of maximum likelihood Markov model

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

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

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

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

Advantages and Disadvantages of MC vs. TD (3)

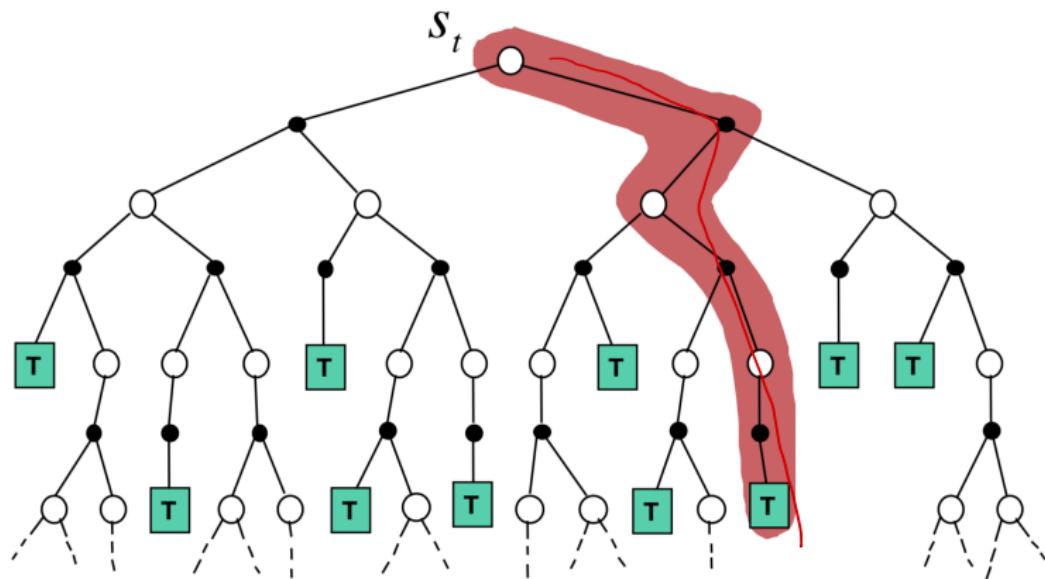
- TD exploits Markov property
 - ▶ Usually more efficient in Markov environments
- MC does not exploit Markov property
 - ▶ Usually more effective in non-Markov environments
- MC has lower error on past data, but higher error on future data

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

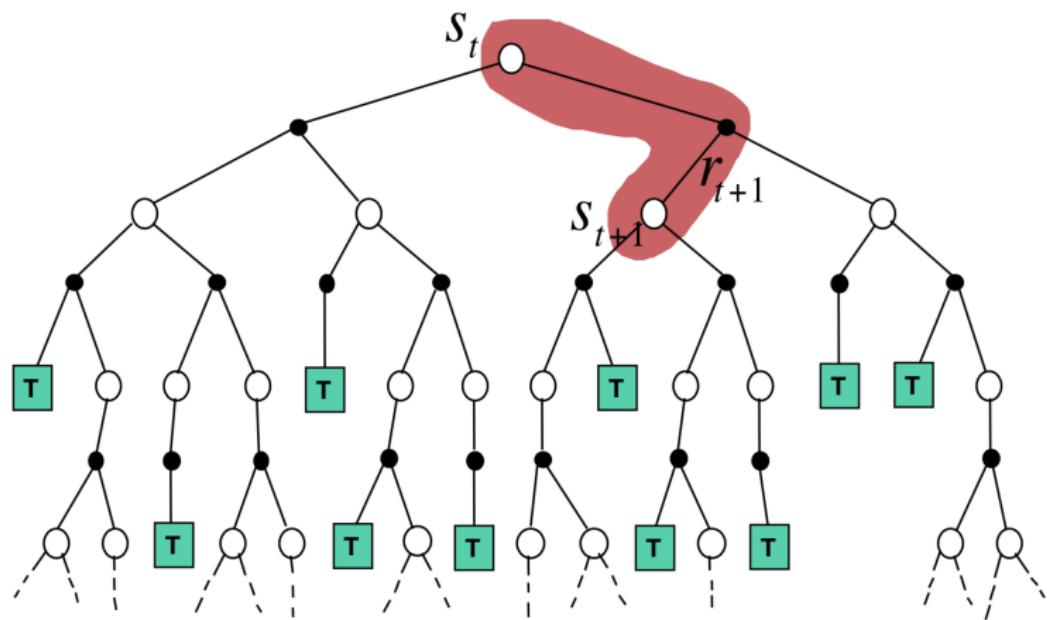
Monte-Carlo Backup

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



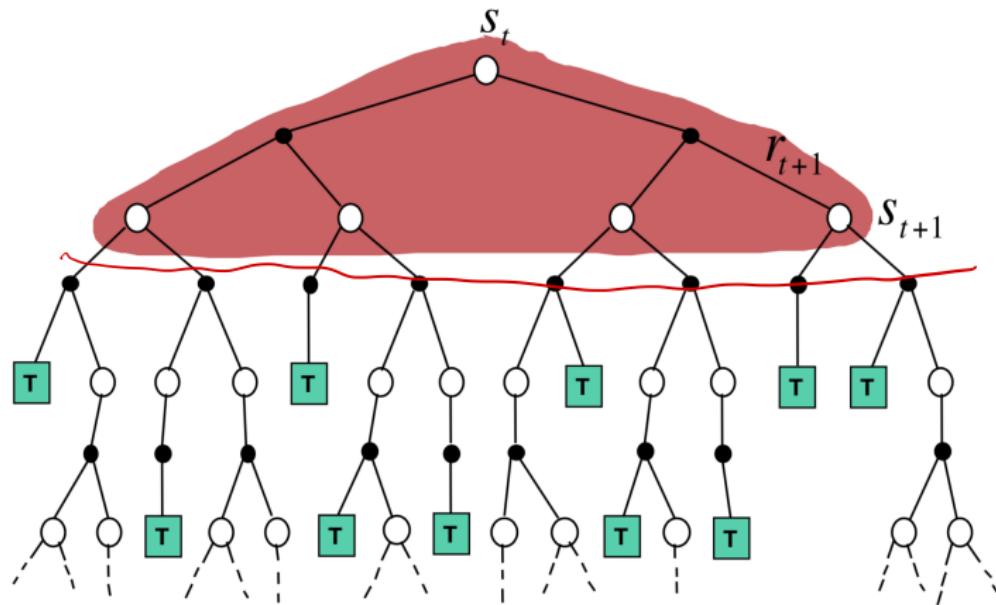
Temporal-Difference Backup

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



Dynamic Programming Backup

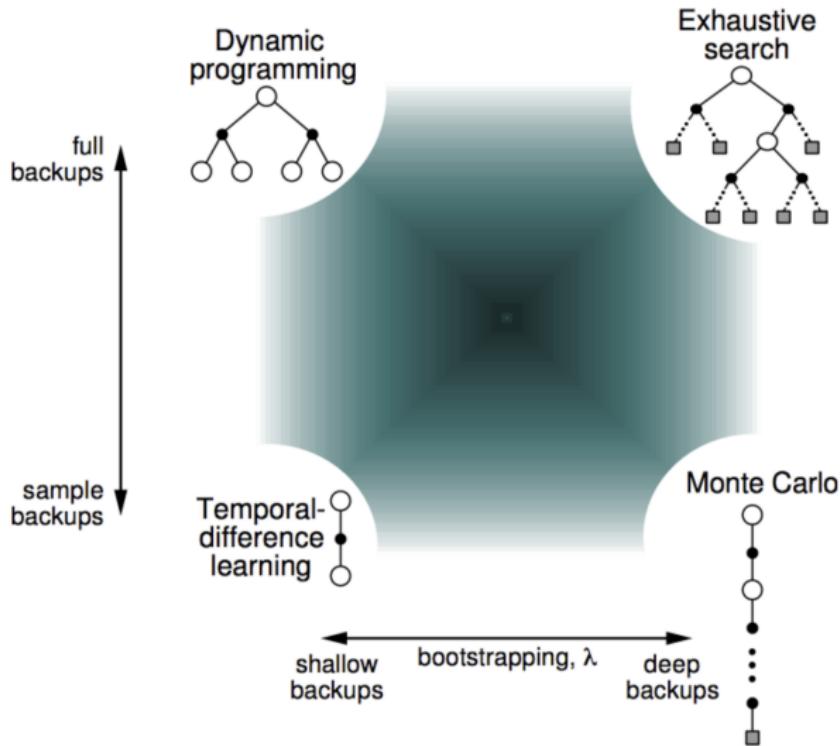
$$V(S_t) \leftarrow \mathbb{E}_{\pi}[R_{t+1} + \gamma V(S_{t+1})]$$



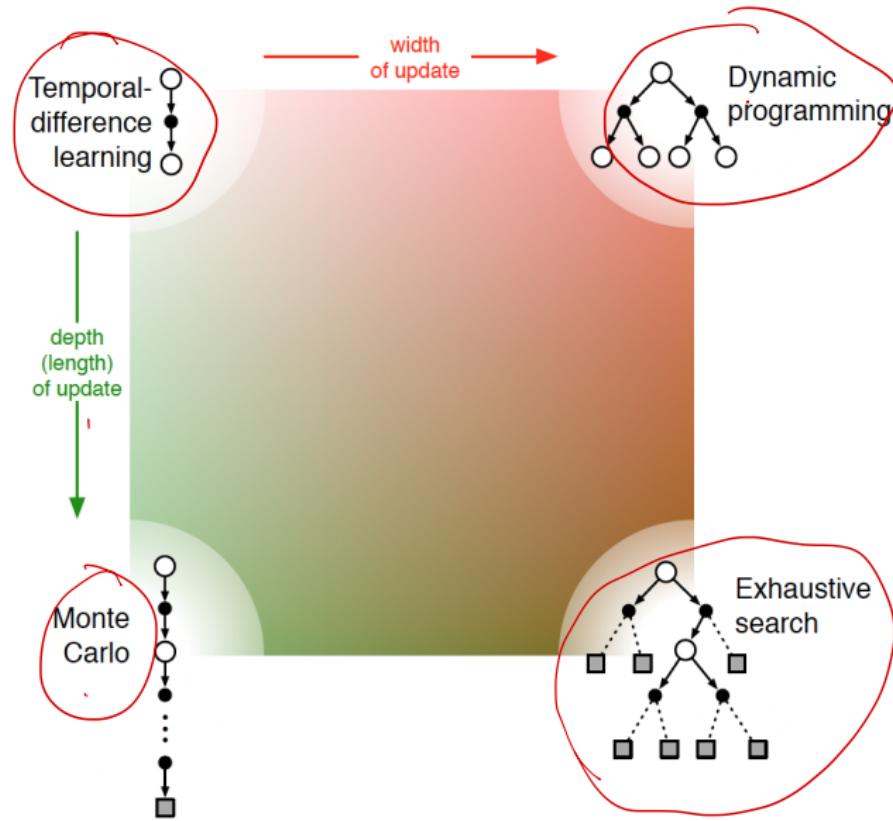
Bootstrapping and Sampling

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

Unified View of Reinforcement Learning



Unified View of Reinforcement Learning

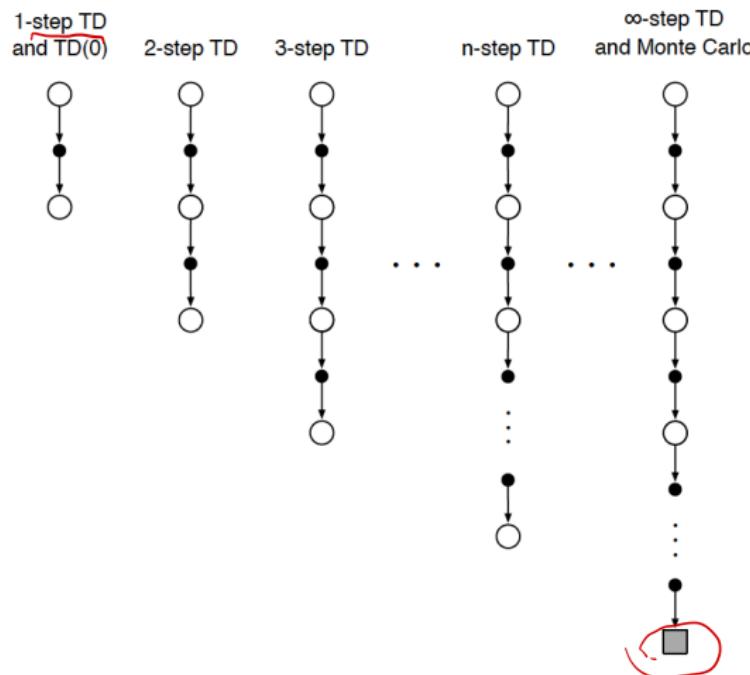


Outline

- 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 $n = 1, 2, \infty$:

$$\underbrace{n=1}_{\text{TD}}$$

$$G_t^{(1)} = \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD}}$$

$$\underbrace{n=2}_{\text{TD}}$$

$$G_t^{(2)} = \underbrace{R_{t+1} + \gamma R_{t+2}}_{\text{TD}} + \underbrace{\gamma^2 V(S_{t+2})}_{\text{TD}}$$

⋮

⋮

$$\underbrace{n=\infty}_{\text{MC}}$$

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

- Define the n -step return

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

- n -step temporal-difference learning

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

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

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

- 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 $S_0 \neq \text{terminal}$

$$T \leftarrow \infty$$

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 $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)

If $\tau > 0$:

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$$

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 $\tau \equiv T - 1$

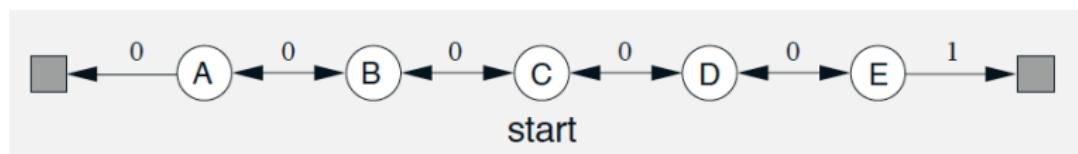
Error Reduction Property

- 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

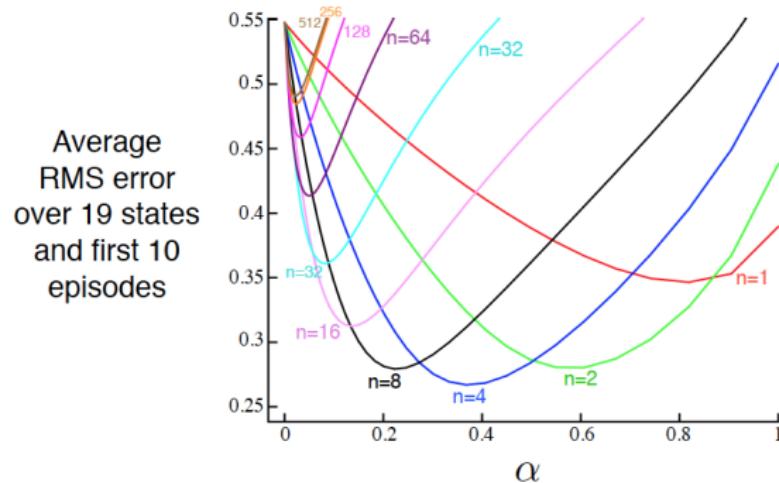
Random Walk Examples



- How does 2-step TD work here?
- How about 3-step TD?

Large Random Walk Example

- n-step TD for 19-state random walk
- An intermediate α is the best
- An intermediate n is the best



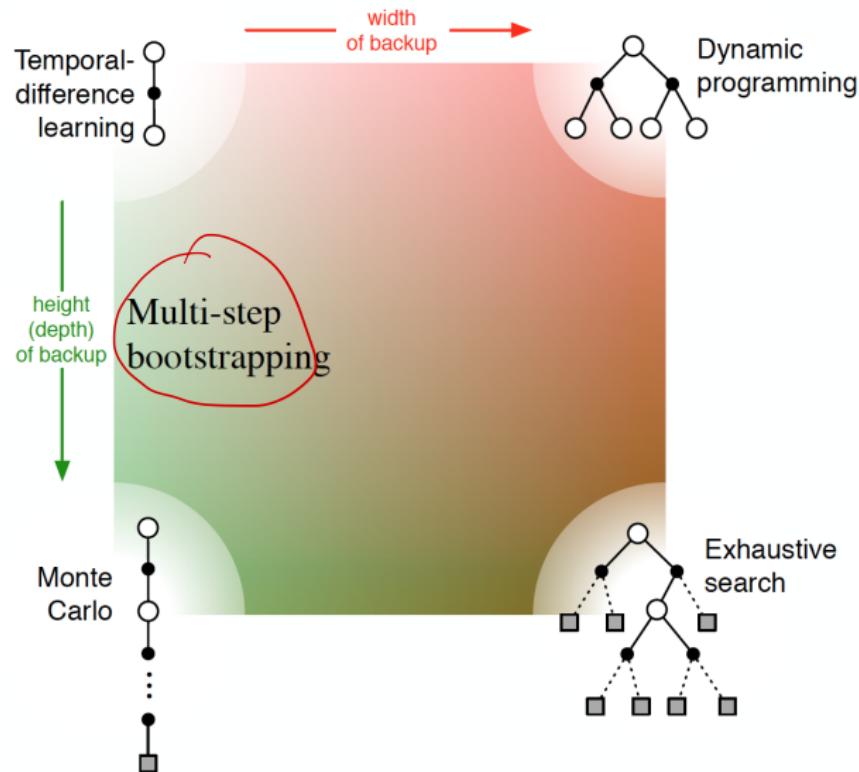
Summary of n-step TD Methods

- 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 $\text{TD}(\lambda)$
- 6 References

Unified View



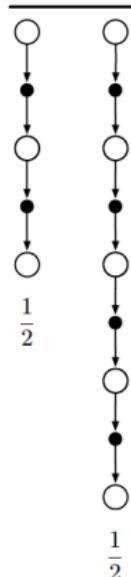
Averaging n -Step Returns

- We can average $\underline{n\text{-step returns}}$ over $\underline{\text{different } n}$
- e.g. average the $\underline{2\text{-step}}$ and $\underline{4\text{-step}}$ returns

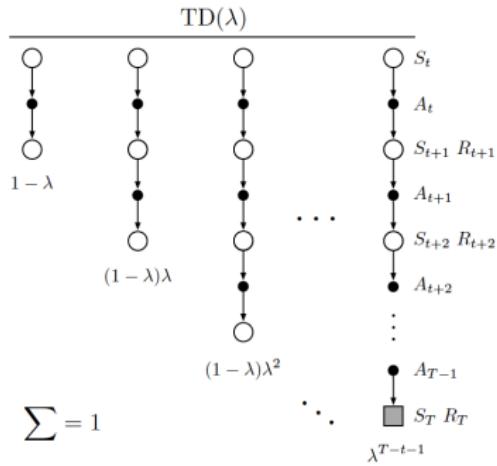
$$\frac{1}{2} \underline{G^{(2)}} + \frac{1}{2} \underline{G^{(4)}}$$

- Combines information from $\underline{\text{two different time-steps}}$
- Can we efficiently combine information from all time-steps?

$TD(\lambda)$



λ -return



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

λ

$$= (1-\lambda^n) \Big|_{n=\infty}$$

$$= 1$$

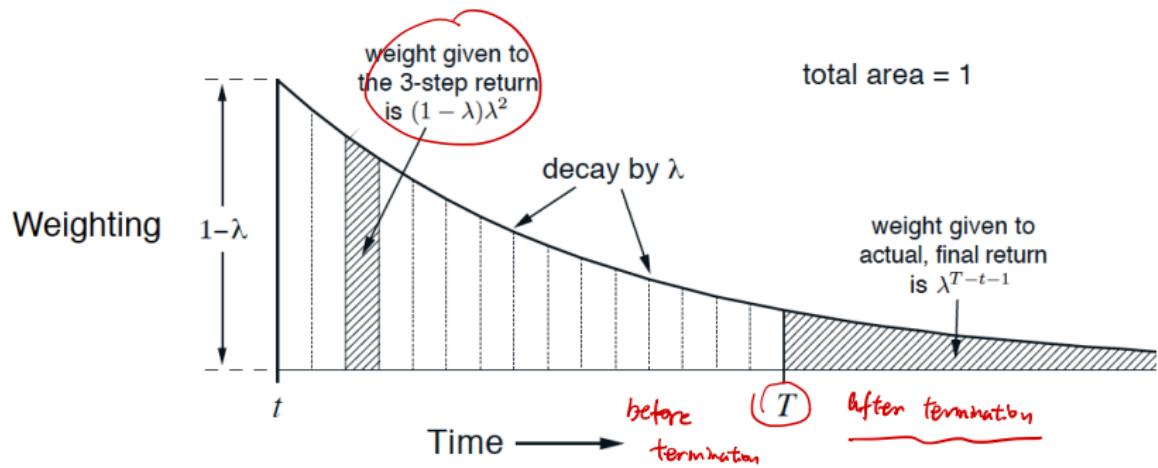
- The λ -return G_t^λ combines all n -step returns $G_t^{(n)}$
 - Using weight $(1 - \lambda)\lambda^{n-1}$

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

- Forward-view TD(λ)

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

TD(λ) Weighting Function



$$\begin{aligned} G_t^\lambda &= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \\ &= \underbrace{(1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)}}_{\text{before termination}} + \underbrace{\lambda^{T-t-1} G_t}_{\text{after termination}} \end{aligned}$$

Relation to TD(0) & MC

- The λ -return can be rewritten as:

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

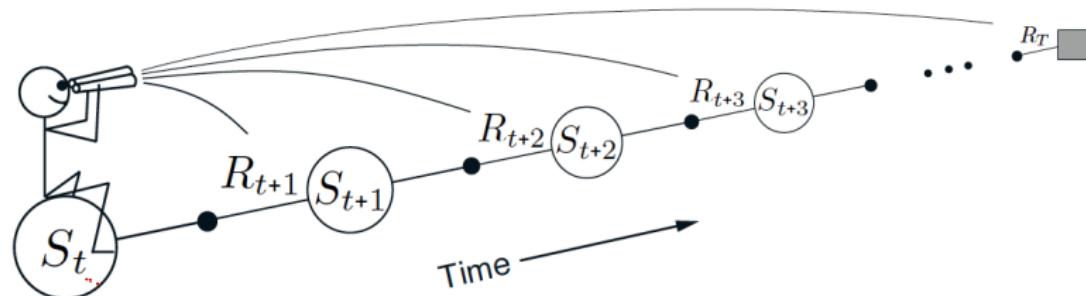
- if $\lambda = 1$, you get the MC target:

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

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

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

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 for computing
- Update online, every step, from incomplete sequences

Eligibility Traces

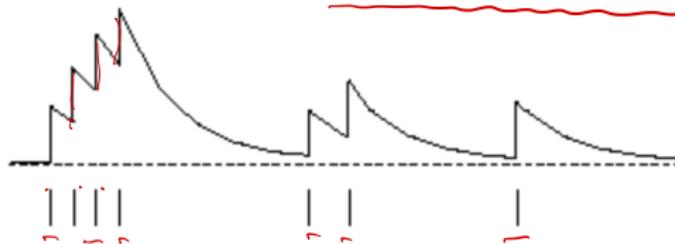


- Credit assignment problem: did bell or light cause shock?
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$E_t(s) = \begin{cases} r > E_{t-1}(s) & \text{if } S_t = s \\ r > E_{t-1}(s) + 1 & \text{if } S_t \neq s \end{cases}$$

$$\underline{E_0(s) = 0}$$

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



Accumulating eligibility trace

Azuo

times of visits to a state

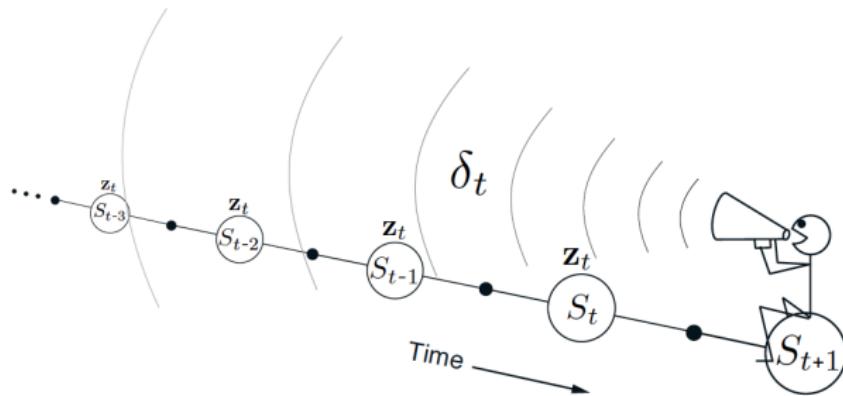
Eligibility Traces

- Eligibility trace keep track of the last time we visited a particular state
- The current reward is then assigned to recently visited states.
- States that have not been visited for a long time are not given much credit for the current reward.
- The eligibility trace for each state decays as $\gamma\lambda$
- The eligibility trace for the state just visited is increased by 1

$$E_t(s) = \gamma\lambda E_{t-1}(s) \quad \text{if } S_t \neq s$$

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

Backward View TD(λ)



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

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

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

Step-size

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

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

TD(λ) and MC

- When $\lambda = 1$, 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

Theorem

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

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

Summary of TD(λ) algorithms

- Another way of interpolating between MC and TD methods
- A way of implementing compound λ -return targets
- A basic mechanistic idea: a short-term, fading memory E.T.
- A new style of algorithm development & analysis
 - ▶ the forward-view & backward-view transformation
 - ▶ Forward view: conceptually simple, good for theory & intuition
 - ▶ Backward view: computationally congenial implementation of the forward view

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.