Model Free Prediction Methods:

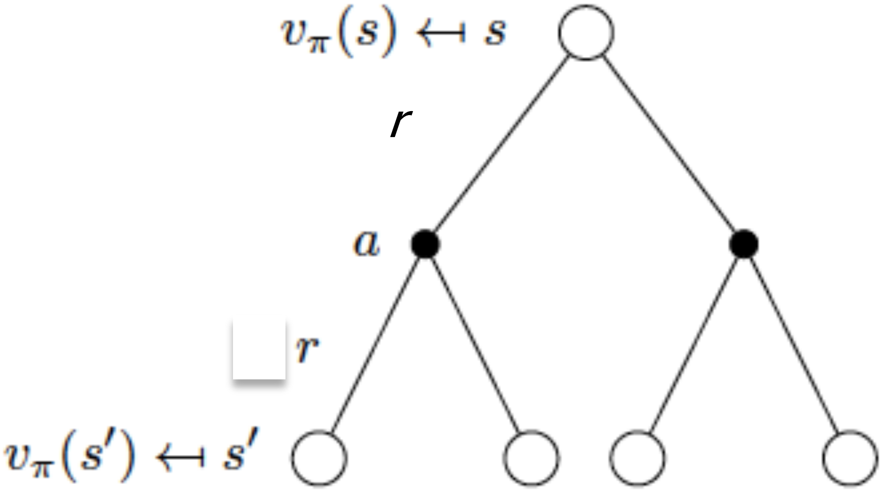
Monte Carlo and Temporal Difference Algorithms

Lecture 4

Subir Varma

Bellman Expectation Equation for v_π

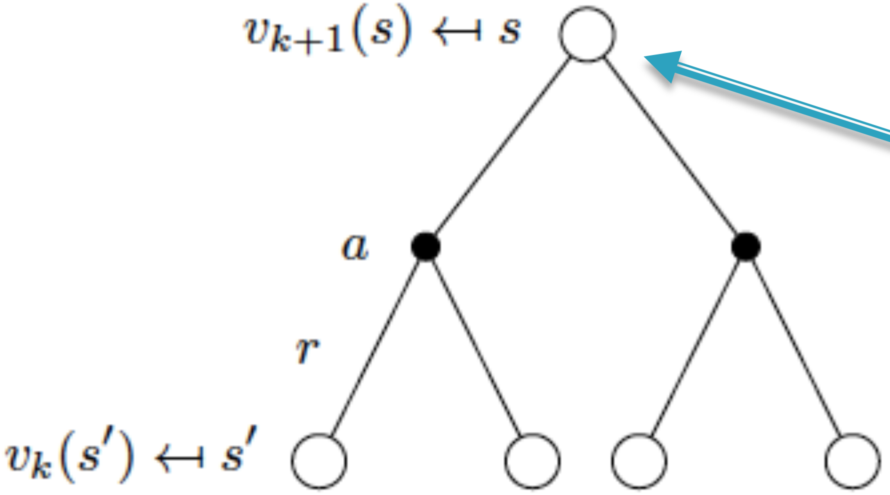
Principle of Optimality
Decompose the problem into:
(1) A smaller problem that is easy to solve, and
(2) A bigger problem, that is assumed to be solved
(3) Put the 2 parts together to solve original problem



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Value Function for State s = One Step Reward +Value Function for Next State s'

Iterative Policy Evaluation



Every state gets its turn being the root

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

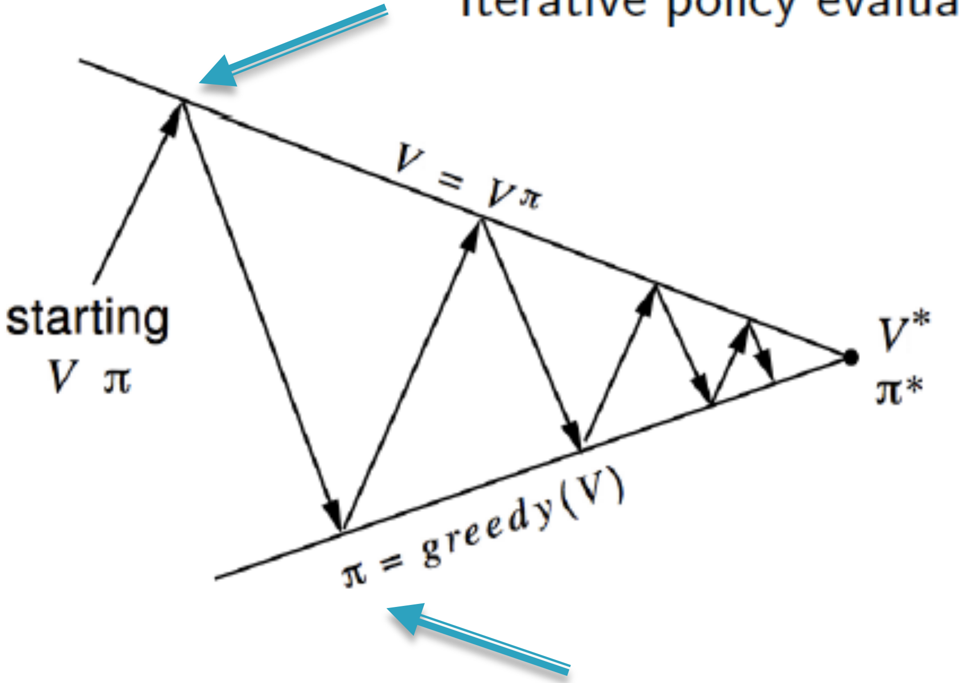
$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

Bellman Expectation Equation

Policy Iteration – Find Best Policy

Policy evaluation Estimate v_π

Iterative policy evaluation



This process is guaranteed to converge to optimal Value Function V* and thus the optimal policy

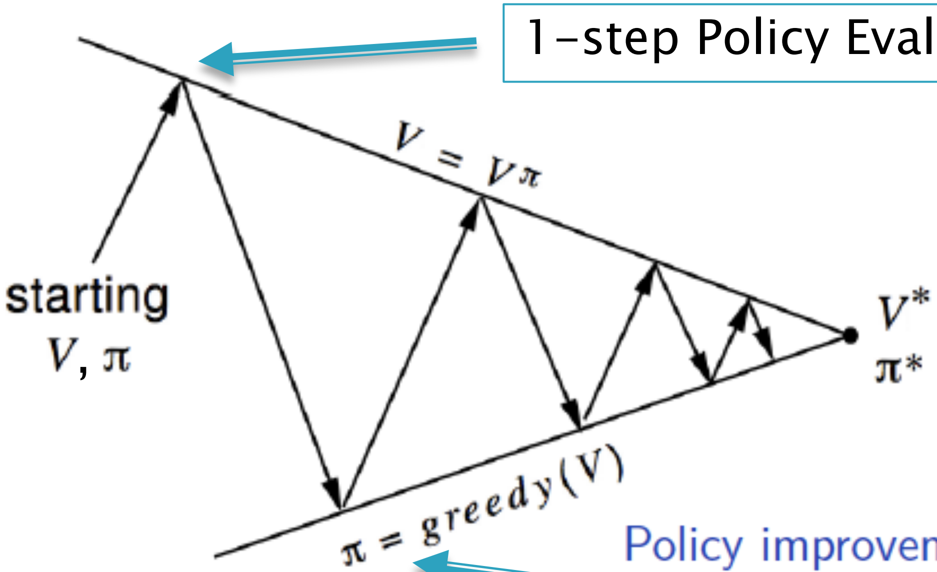
Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Generalized Policy Iteration

Find Heuristics to be able
to solve Problems with huge
number of states and/or actions

1-step Policy Evaluation

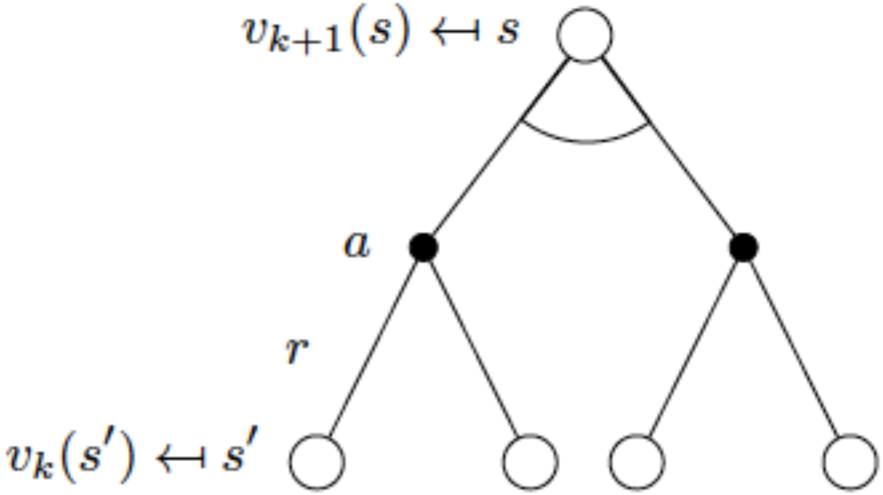


Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

- Iterate only a few times, even just once (k = 1)
- Don't have to update all the states in each iteration update only those that are actually visited

Value Iteration – Find Best Policy



Turn the Bellman Optimality Equation into an Iterative Update

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

Finding the Optimal Policy

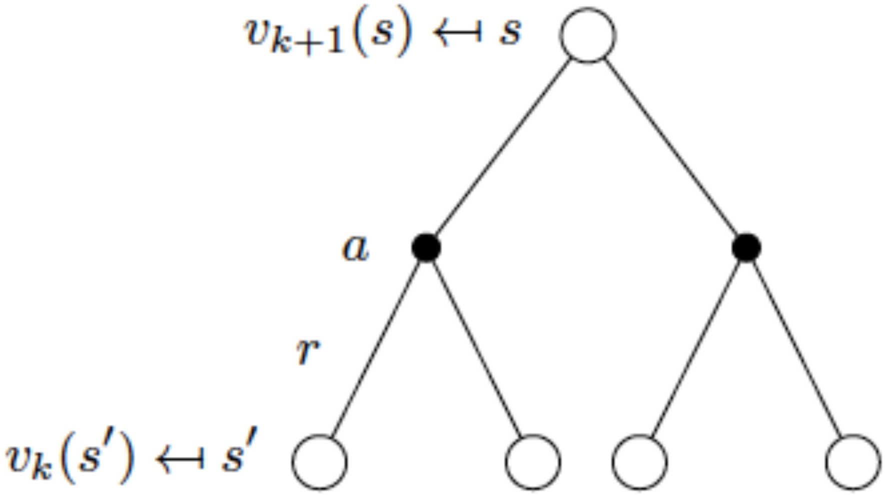
$$v_*(s) = \max_a \left[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right]$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$\boldsymbol{\pi_* = argmax_a(q_*(s, a))}$$

But…

These algorithms are dependent on the knowledge of the MDP Model P



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}^{\pi} \mathbf{v}^k$$

Motivation

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

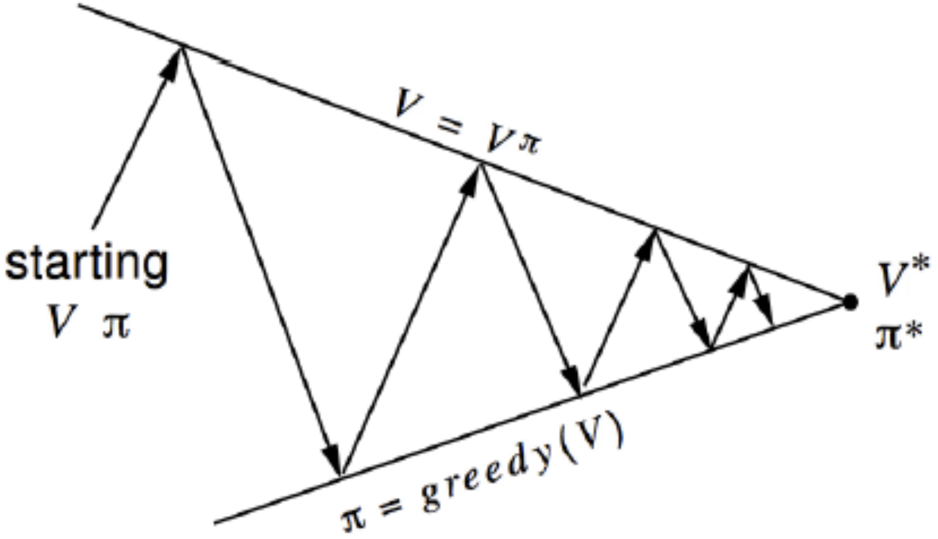
$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}^{\pi} \mathbf{v}^{k}$$

- Policy Iteration and Value Iteration Algorithms don’t work if:
  - The Environment Model is not known, or
  - The number of states is extremely large

How can we find the Value Function and the Optimal Policy under these conditions

Solution: Instead of Computing these functions from a Model, Learn them from <u>Experience</u>!

Motivation (cont)



Still Doing Policy Iteration

- Experience: Sample sequences of States, Actions and Rewards
  (S, A, R, S’)
- The Experience can be either Real or Simulated

Model Free Reinforcement Learning

- Model for the Environment not known
- Agent uses its interaction with environment to figure out its Value Function and Optimal Policy

- This Lecture: Given a Policy, how do we figure out the Value Function, without knowing the model
- Next lecture: How to find Optimal Policy, without knowing the model

This Lecture

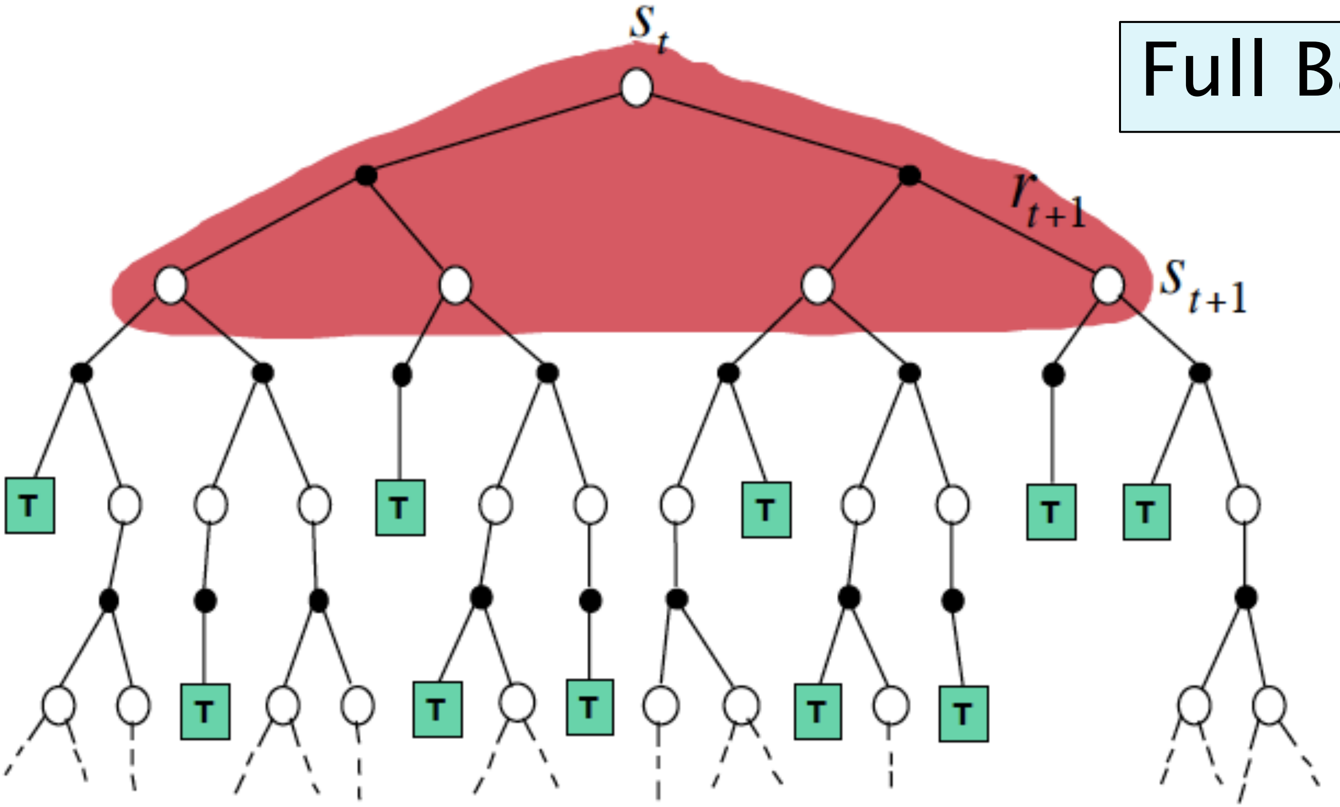
1. Monte Carlo (MC) Learning
   - Look at complete trajectories and estimate the value by looking at sample returns
2. Temporal Difference (TD) Learning
   - Look one step ahead and estimate the return
   - Can be significantly more efficient than MC Learning in practice.
3. TD(n) and TD(λ)
   - Unifies the MC and TD approaches

Monte Carlo Reinforcement Learning

Dynamic–Programming Backup

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_k(s')\right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

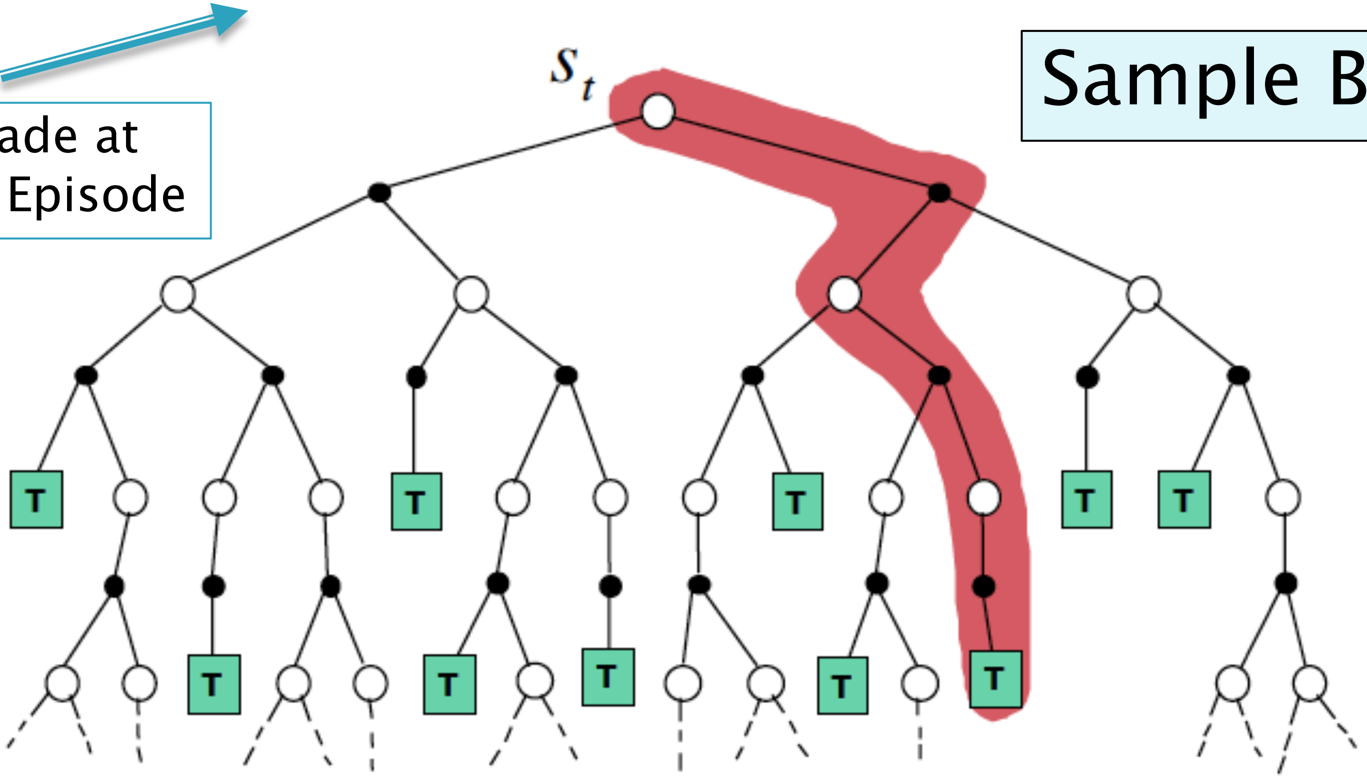


Full Backup

Monte Carlo Backup

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

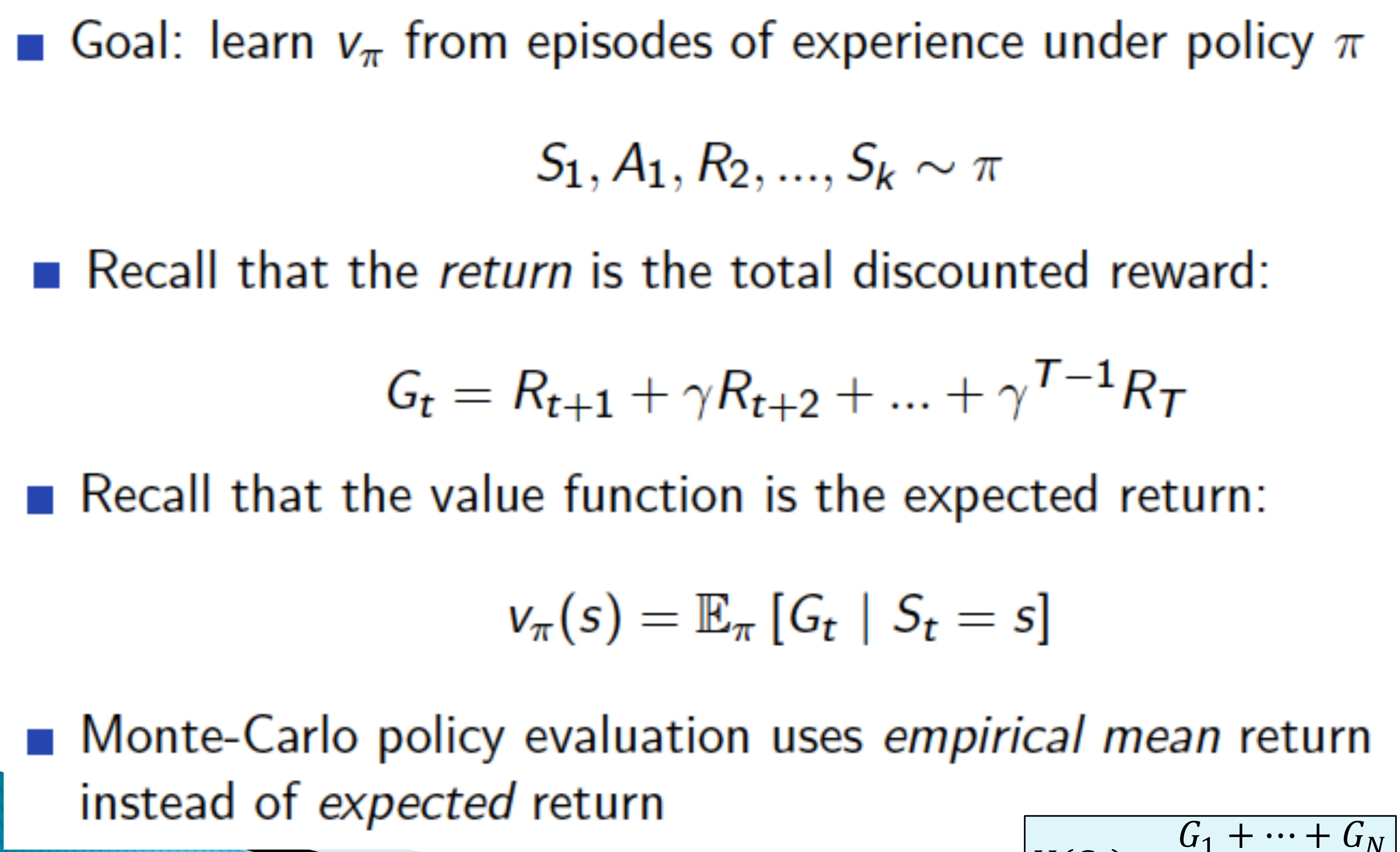
Update made at end of an Episode



Sample Backup

Don’t Need the Model Anymore!

Monte Carlo Policy Evaluation



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

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

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

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

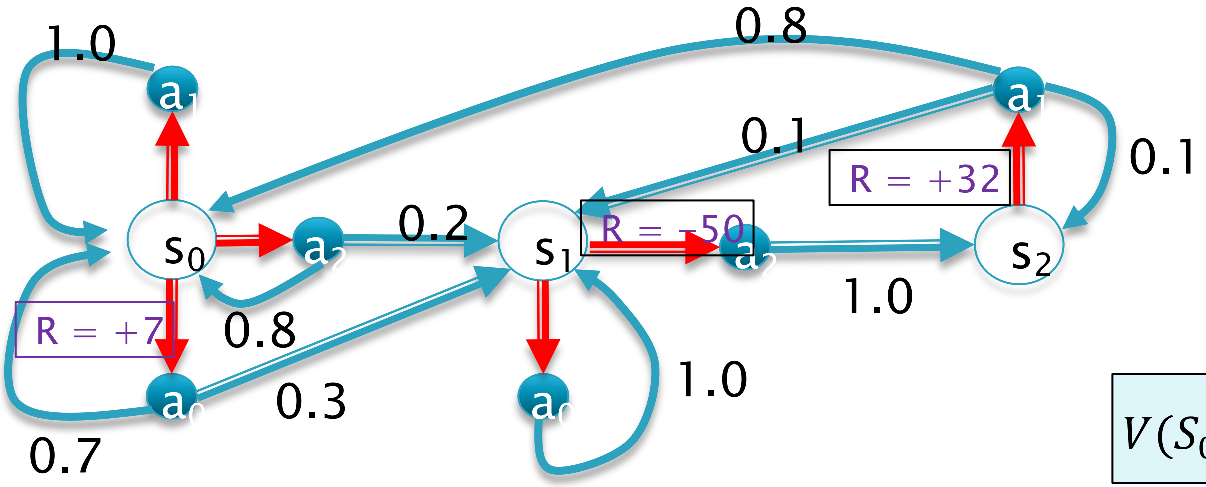
- Recall that the value function is the expected return:

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

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

$$\boxed{V(S_0) = \frac{G_1 + \cdots + G_N}{N}}$$

Example of MC: MDP Returns

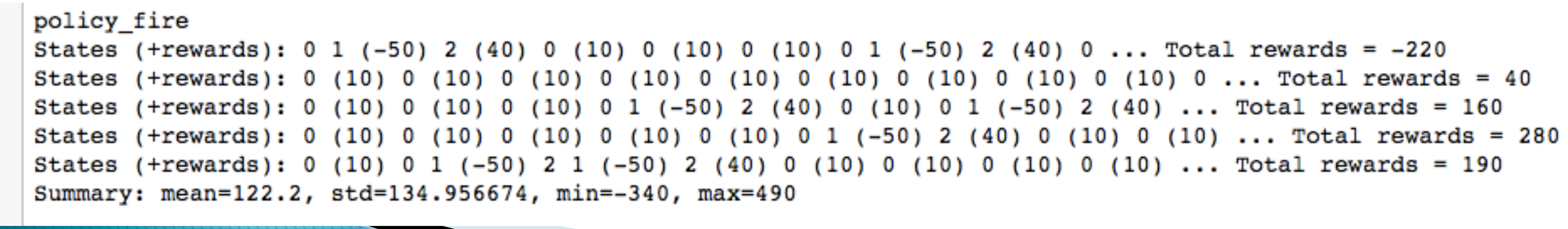


$$V(S_0) = \frac{G_1 + \cdots + G_N}{N}$$

Sample returns, starting from state s$_0$ and $\gamma = 1$,

Average return computed from 1000 episodes and 100 steps per episode

$$G_1 = R_2 + \gamma R_3 + \ldots + \gamma^{T-2} R_T$$



```
policy_fire
States (+rewards): 0 1 (-50) 2 (40) 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 ... Total rewards = -220
States (+rewards): 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 ... Total rewards = 40
States (+rewards): 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 (10) 0 1 (-50) 2 (40) ... Total rewards = 160
States (+rewards): 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 (10) 0 (10) ... Total rewards = 280
States (+rewards): 0 (10) 0 1 (-50) 2 1 (-50) 2 (40) 0 (10) 0 (10) 0 (10) 0 (10) ... Total rewards = 190
Summary: mean=122.2, std=134.956674, min=-340, max=490
```

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

- Caveat: can only apply MC to *episodic* MDPs
  - All episodes must terminate

Computing the Empirical Mean

Two Techniques

- First Visit Monte Carlo
- Every Visit Monte Carlo

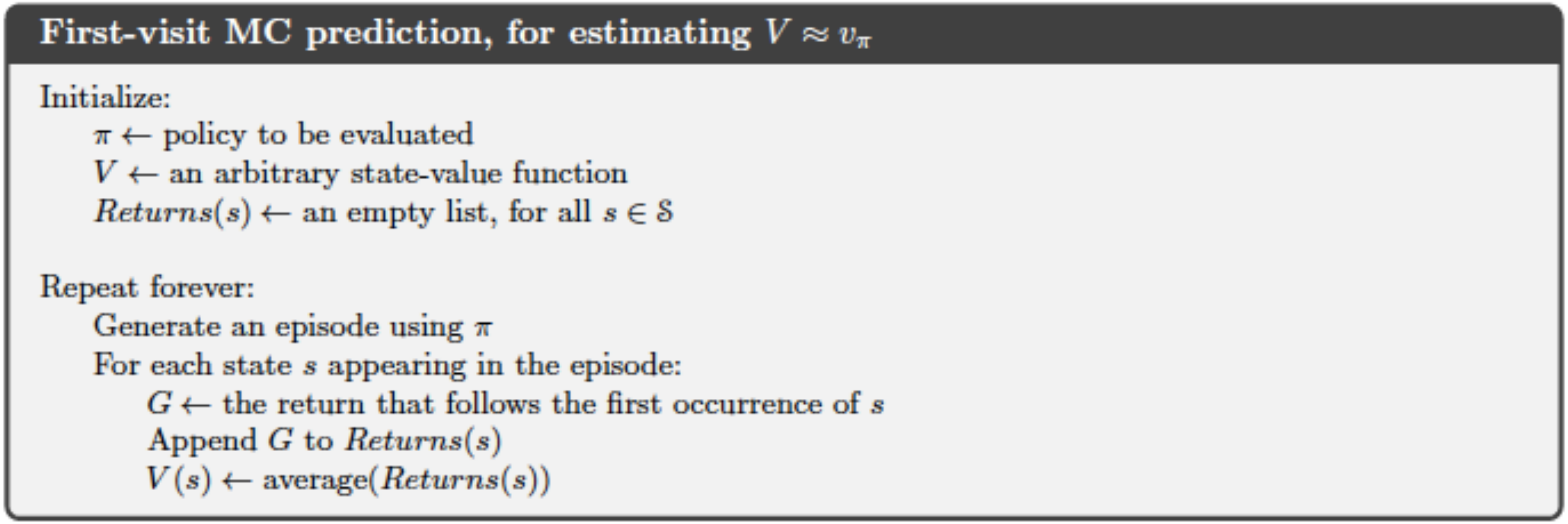
First Visit Monte Carlo Policy Evaluation

- 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 $N(s) \rightarrow \infty$

How quickly does it converge: Variance of error reduces as 1/n

We are sampling instead of doing a full sweep and this breaks the the dependence on the size of the problem state space

First Visit Monte Carlo

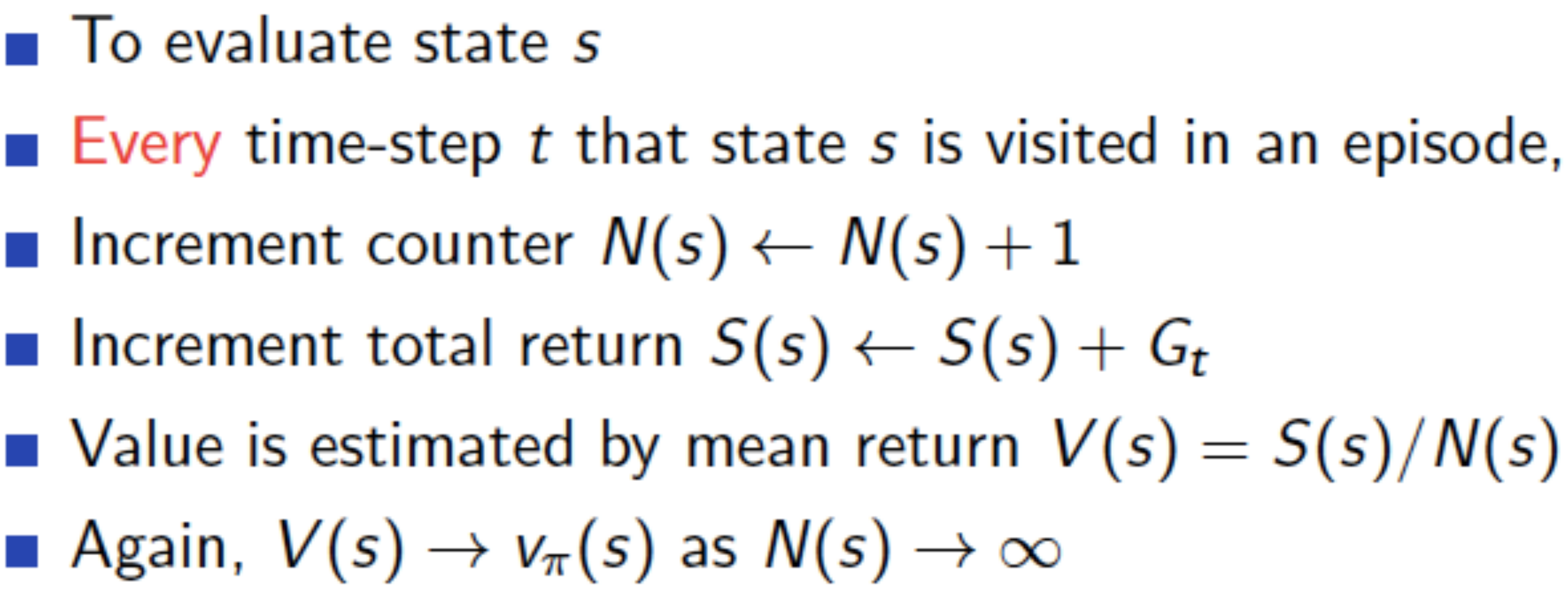


First-visit MC prediction, for estimating $V \approx v_\pi$

```
Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using $\pi$
    For each state $s$ appearing in the episode:
        $G \leftarrow$ the return that follows the first occurrence of $s$
        Append $G$ to $Returns(s)$
        $V(s) \leftarrow$ average($Returns(s)$)
```

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 $N(s) \rightarrow \infty$

In one episode, V(s) can be updated multiple times (for a given s)

Incremental Mean Update

$$V_N = \frac{\sum_{i=1}^{N} G_i}{N}$$

$$= \frac{1}{N}\left(G_N + \sum_{i=1}^{N-1} G_i\right)$$

$$= \frac{1}{N}(G_N + (N-1)V_{N-1})$$

$$= V_{N-1} + \frac{1}{N}(G_N - V_{N-1})$$

New Estimate = Current Estimate + Error Term

Incremental Monte Carlo Updates

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

$$N(S_t) \leftarrow N(S_t) + 1$$

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

Exponential Smoothing

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

Smoothing Parameter
Example: $\alpha = 0.1$

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

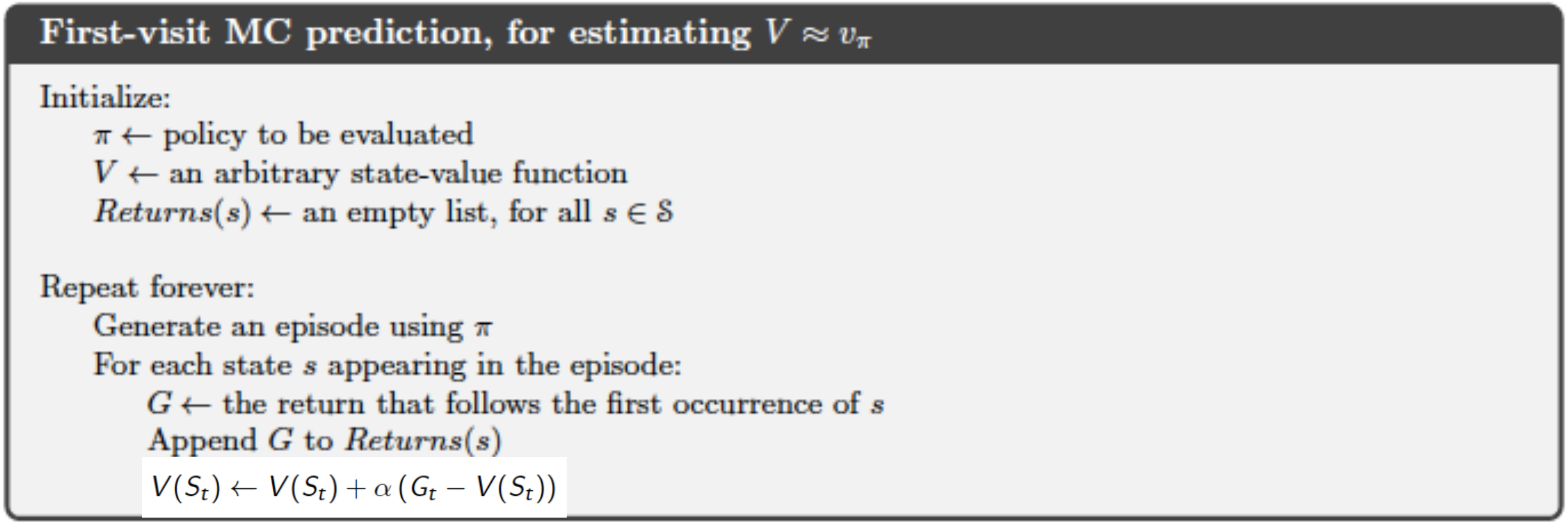
Leads to an exponential forgetting rate

This works better in practice, since with policy improvements the system keeps changing

Sometimes also written as:
$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t$

$$V_{n+1} = \alpha G_1 + \alpha(1-\alpha)^{n-1} G_2 + \cdots + \alpha(1-\alpha)G_n + \alpha G_{n+1}$$

First Visit Monte Carlo with Exponential Smoothing



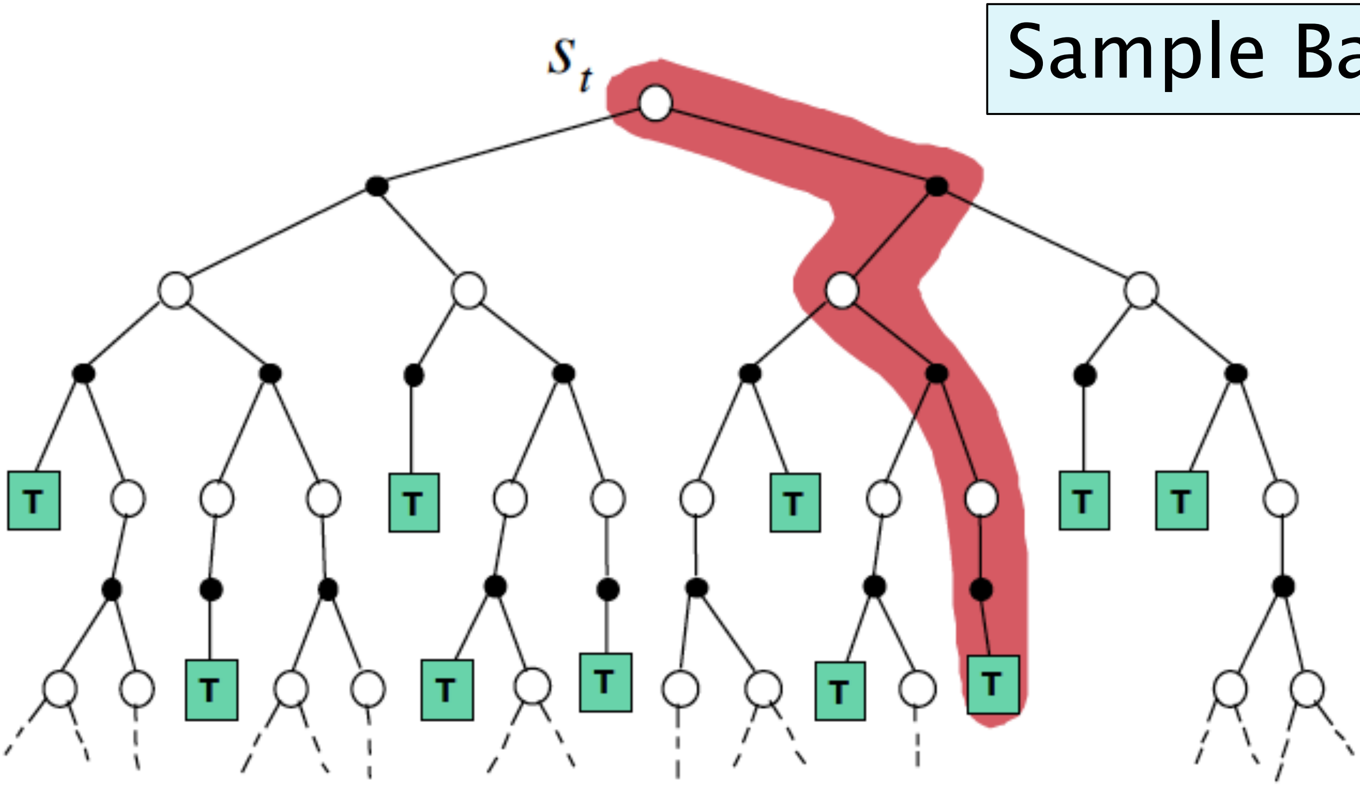
First-visit MC prediction, for estimating $V \approx v_\pi$

```
Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using $\pi$
    For each state $s$ appearing in the episode:
        $G \leftarrow$ the return that follows the first occurrence of $s$
        Append $G$ to $Returns(s)$
        $V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$
```

MC Estimate for a Single State

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



Sample Backup

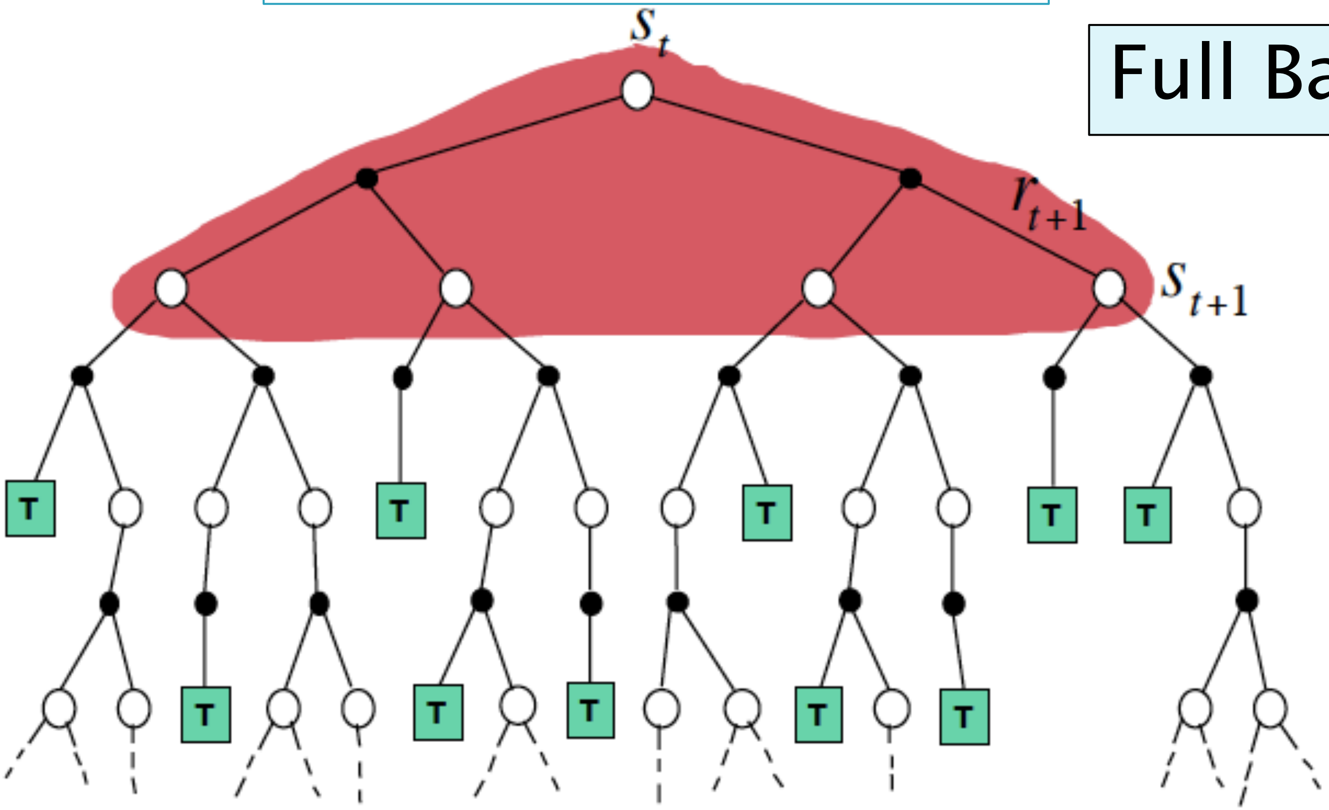
Computational expense of estimating the value of a single state is independent of the number of states

Temporal Difference (TD) Reinforcement Learning

Dynamic-Programming Backup

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_k(s')\right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \mathbf{v}^k$$



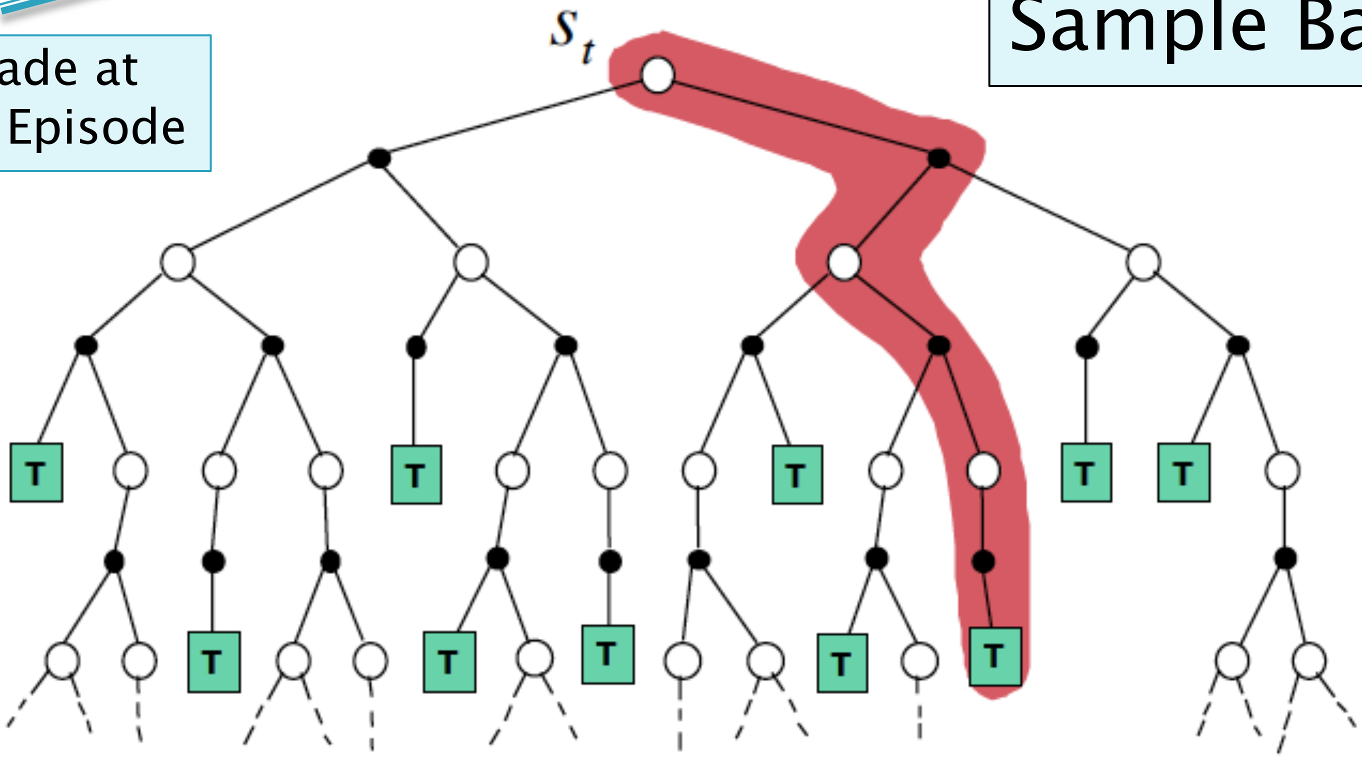
Full Backup

Monte Carlo Backup

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

Update made at end of an Episode

Sample Backup



Issue with MC Learning

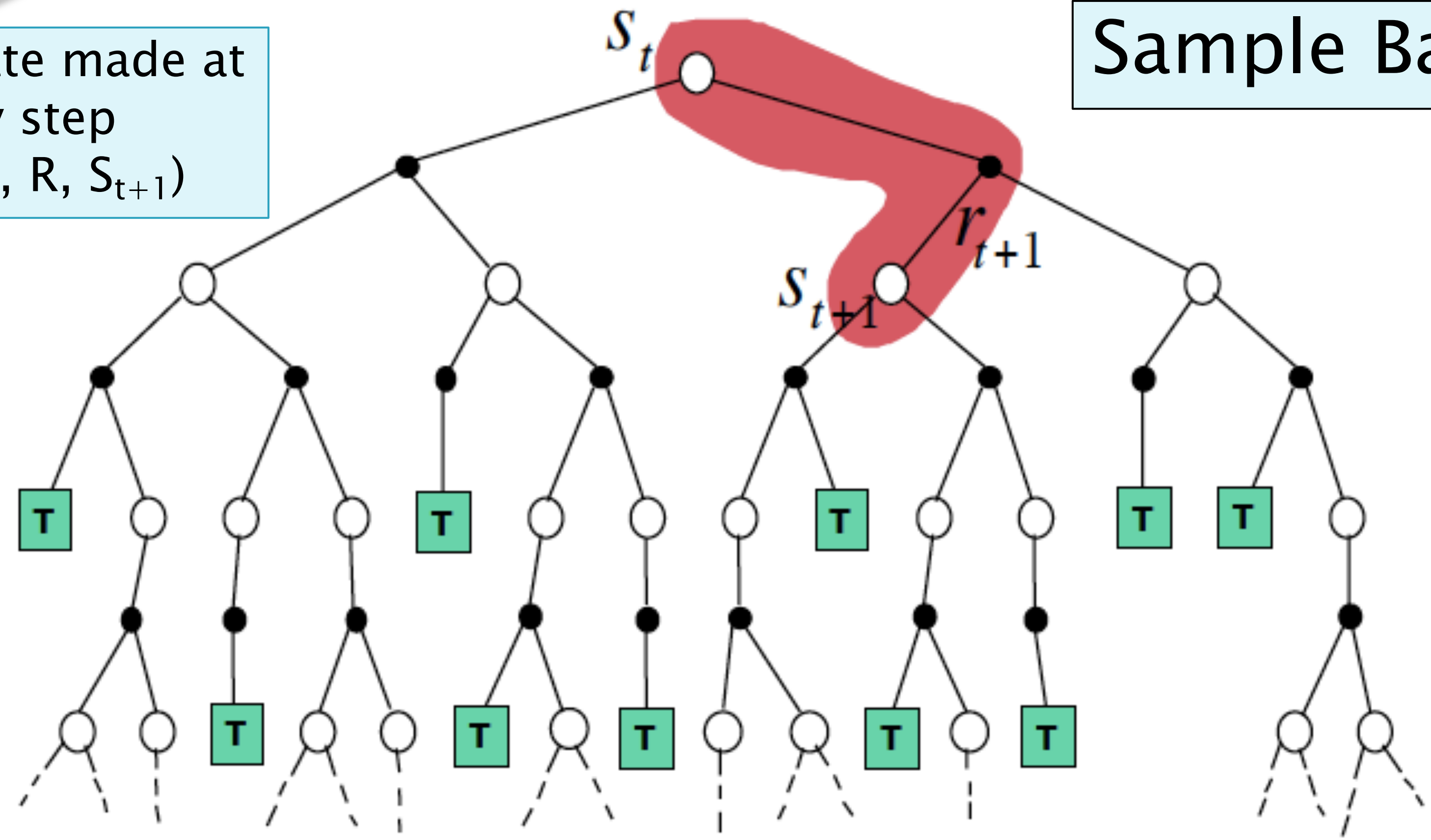
- Having to wait until end of episode to compute update
- Downsides:
  - What if something “bad” happens at end of episode
  - Some MDPs are continuous, i.e., never ending episodes

Temporal-Difference Backup

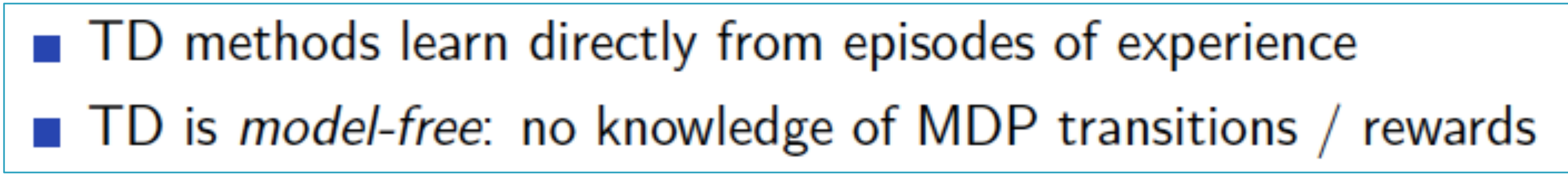
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

Update made at every step (S_t, A, R, S_{t+1})

Sample Backup

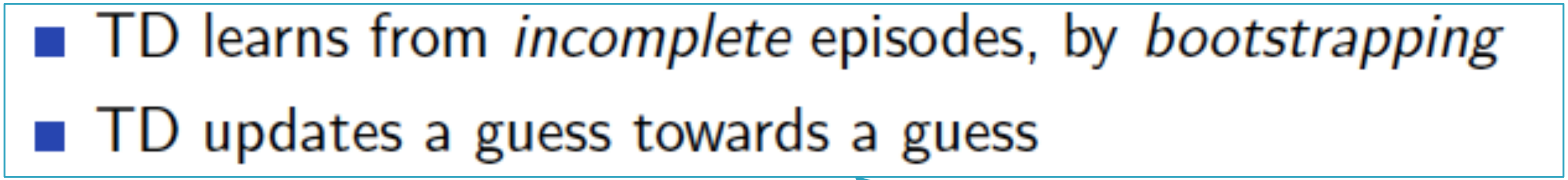


Temporal-Difference (TD) Learning



- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards

Similar to Monte Carlo

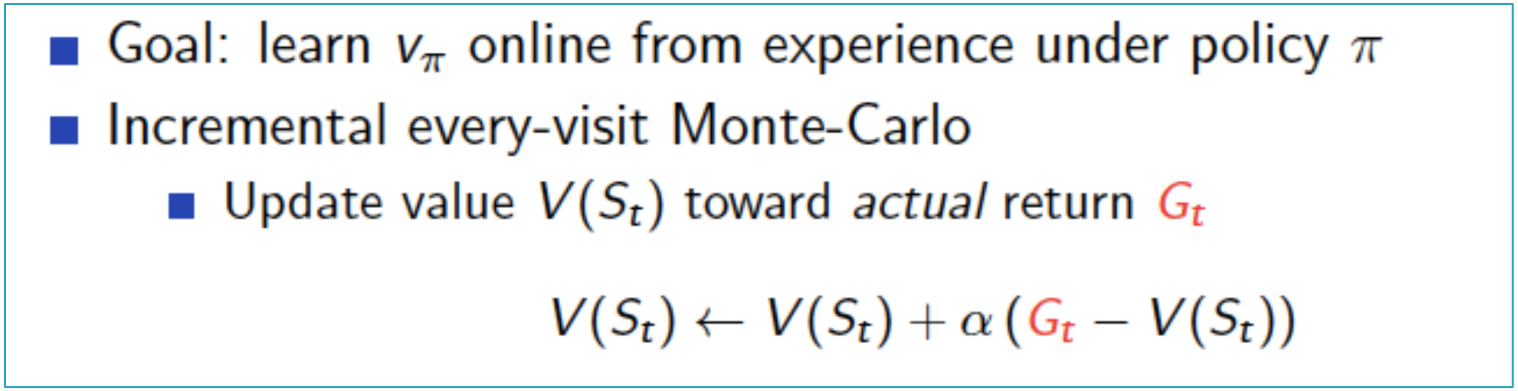


- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

Different from Monte Carlo
Similar to Dynamic Programming

Derivation of TD Update Rule

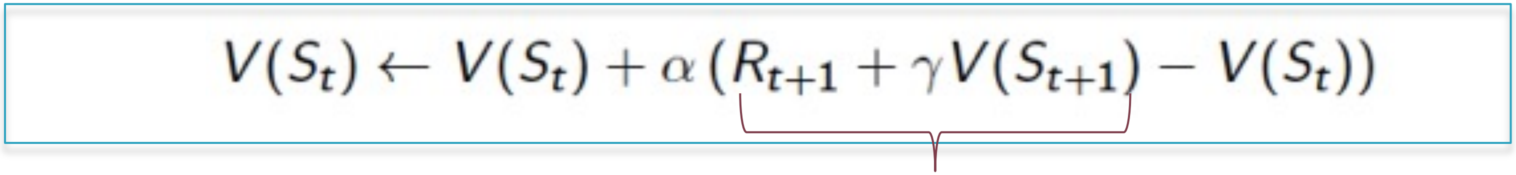
With MC



- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

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

With TD



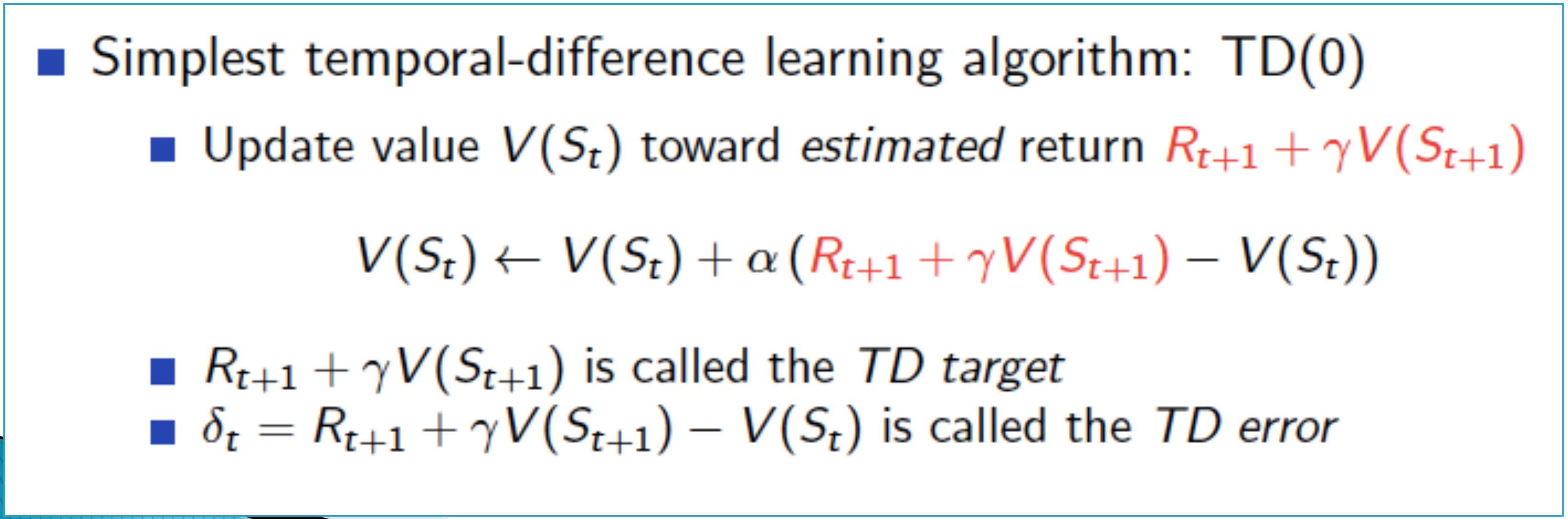
$$V(S_t) \leftarrow V(S_t) + \alpha \left(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Best estimate of } G_t} - V(S_t) \right)$$

TD Update

$$G_t = \underset{\substack{\nearrow \\ \text{1-step} \\ \text{reward}}}{R_{t+1}} + \underbrace{\gamma R_{t+2} + \ldots + \gamma^{T-1} R_T}_{\substack{\text{Estimated Reward for} \\ \text{rest of trajectory}} \; = \; \gamma V(S_{t+1})}$$

G_t: Real Return

$R_{t+1} + \gamma V(S_{t+1})$: Estimated Return

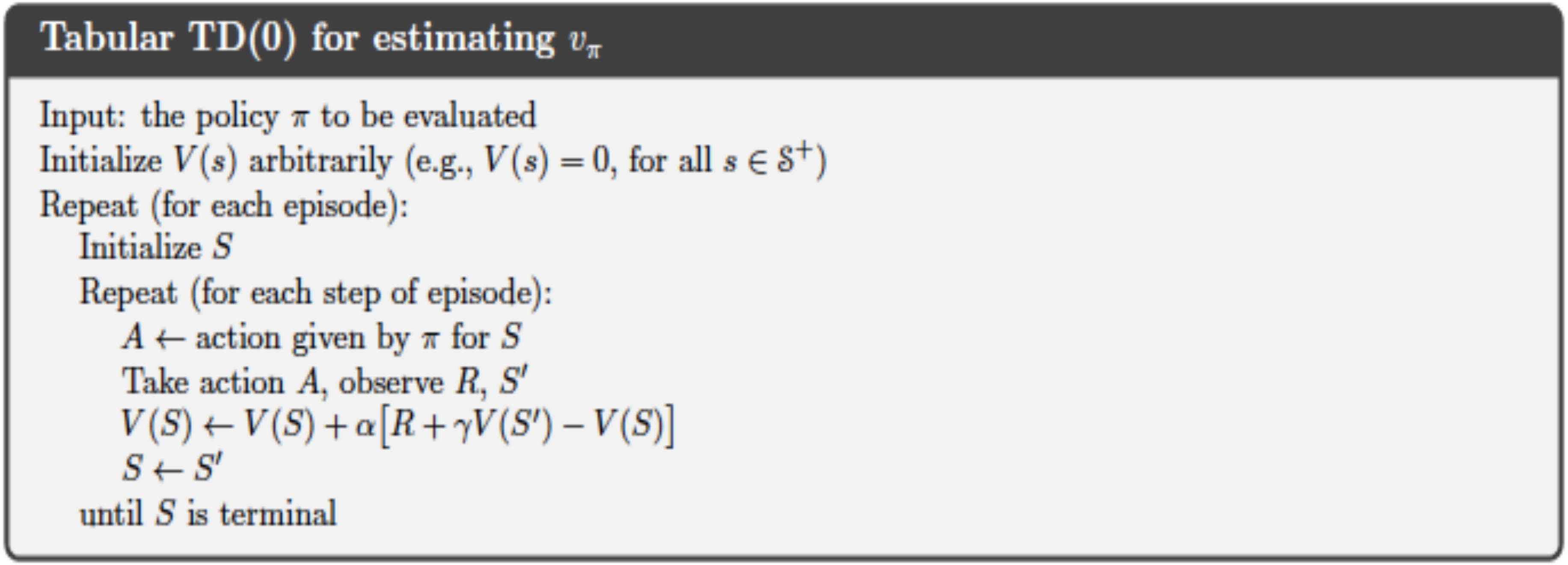


- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

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

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

TD Algorithm



```
Tabular TD(0) for estimating $v_\pi$

Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Sometimes also written as:

$$V(S) \leftarrow (1-\alpha)V(S) + \alpha[R + \gamma V(S')]$$

Advantages and Disadvantages of MC vs TD

- 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

Bias/Variance Trade-Off

Leads to faster convergence

- 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

$$\boxed{V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t}$$

High Variance

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

Lower Variance

$$\boxed{V(S) \leftarrow (1-\alpha)V(S) + \alpha[R + \gamma V(S')]}$$

Bias/Variance Trade-Off

- TD target $R_{t+1} + \gamma V(S_{t+1})$ is *biased* estimate of $v_\pi(S_t)$

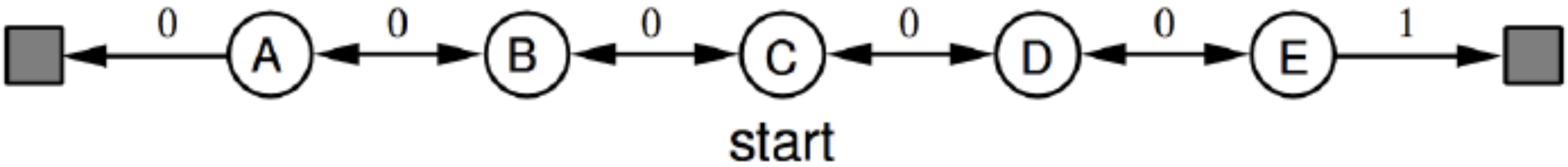
- Return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$ is *unbiased* estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is *unbiased* estimate of $v_\pi(S_t)$

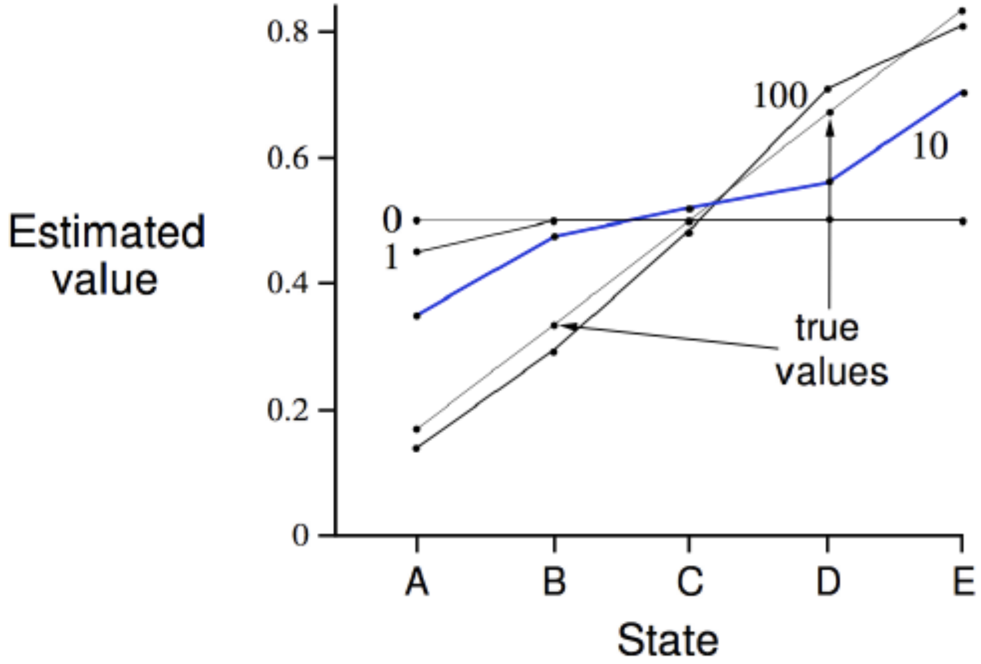
Advantages and Disadvantages of MC vs TD (2)

- 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

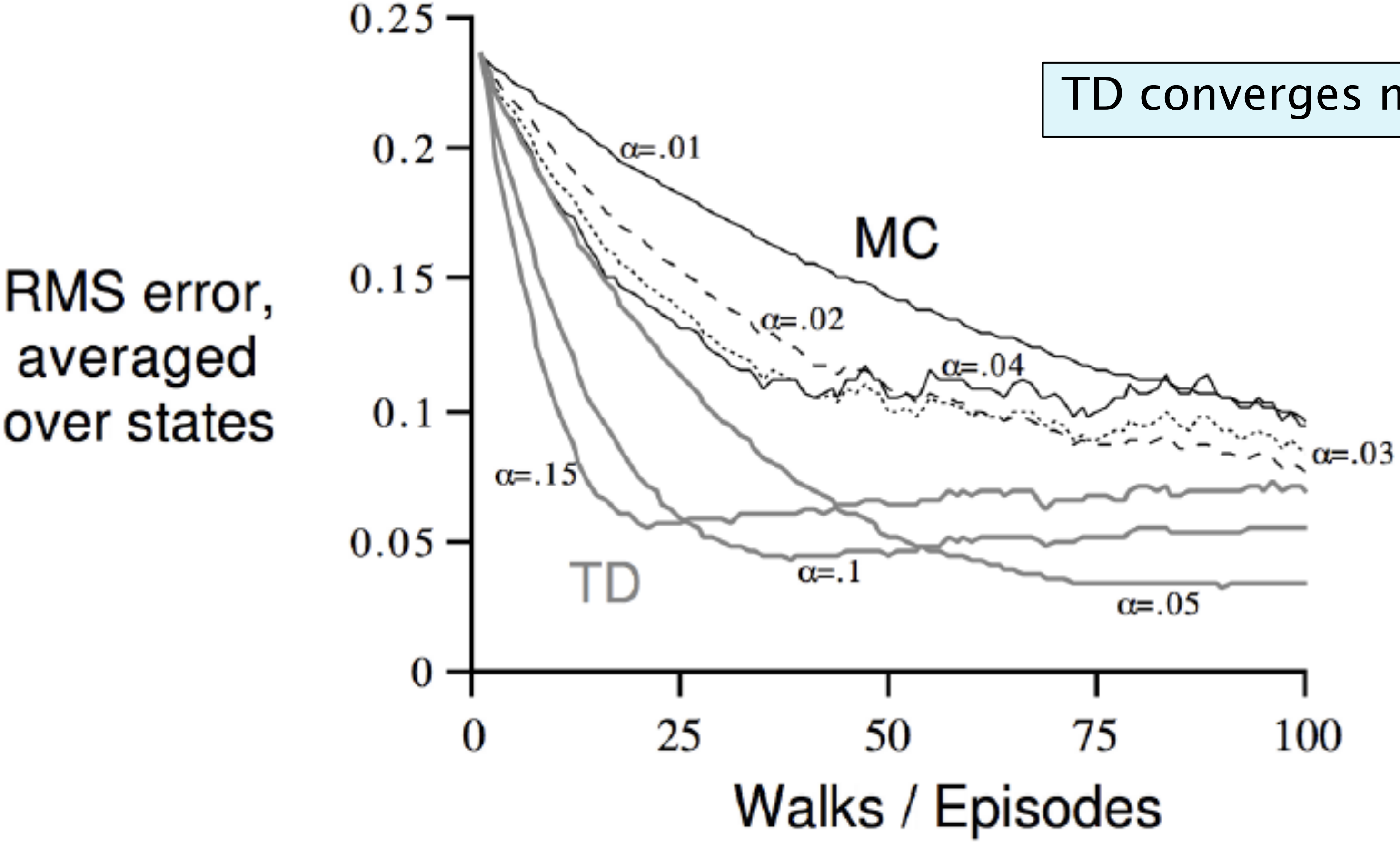
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

Random Walk Example





Random Walk: MC vs TD



TD converges much faster

Larger α: Faster but noisier convergence
Smaller α: Slower and smoother convergence

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?

$$s_1^1, a_1^1, r_2^1, ..., s_{T_1}^1$$

$$\vdots$$

$$s_1^K, a_1^K, r_2^K, ..., s_{T_K}^K$$

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

AB Example

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

$\mathrm{A}, 0, \mathrm{B}, 0$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 0$

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

AB Example

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

A, 0, B, 0

B, 1

B, 1

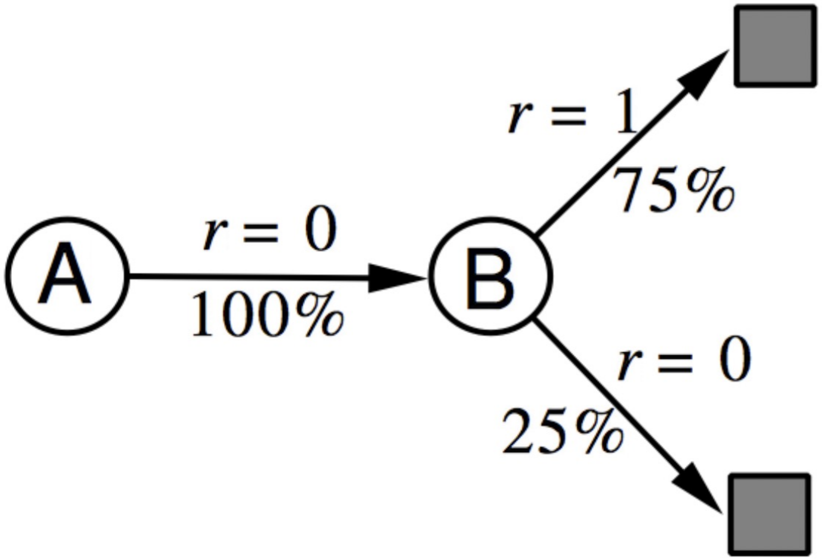
B, 1

B, 1

B, 1

B, 1

B, 0



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

Convergence Properties

- MC converges to solution with minimum mean-squared error
    - Best fit to the observed returns

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

    - In the AB example, $V(A) = 0$
- TD(0) converges to solution of max 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

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

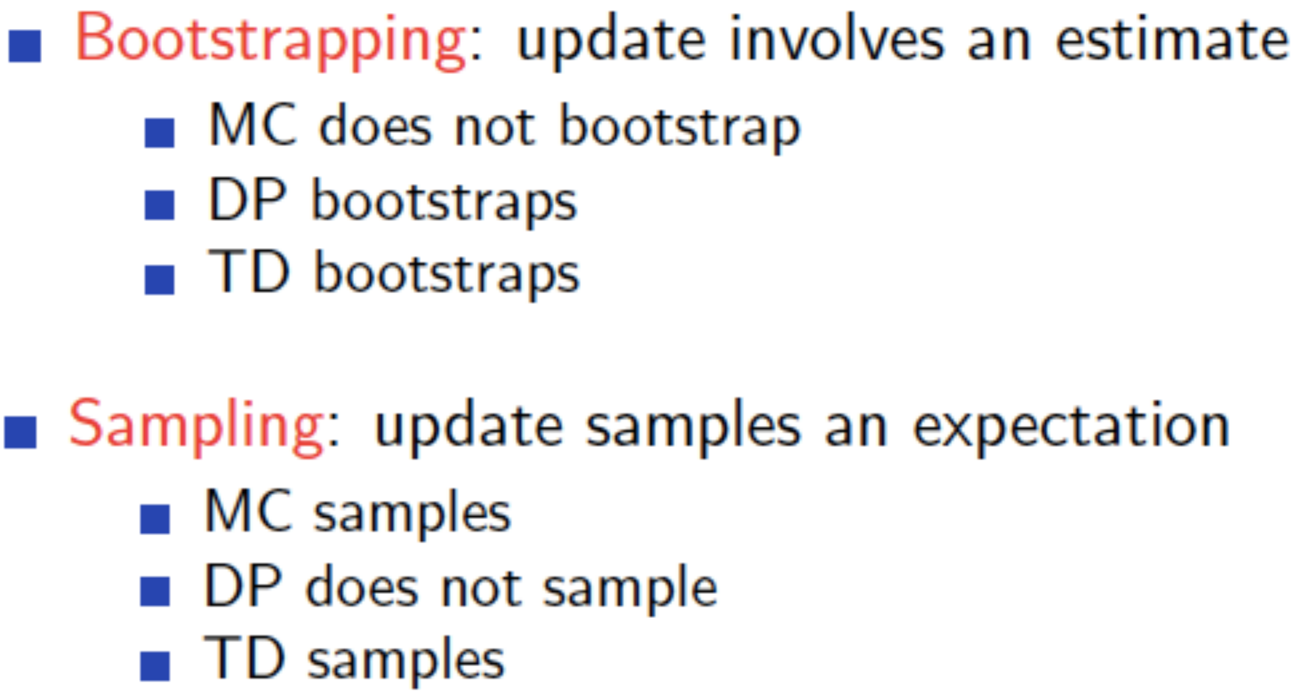
$$\hat{\mathcal{R}}_{s}^{a} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k = s, a) r_t^k$$

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

MC vs TD

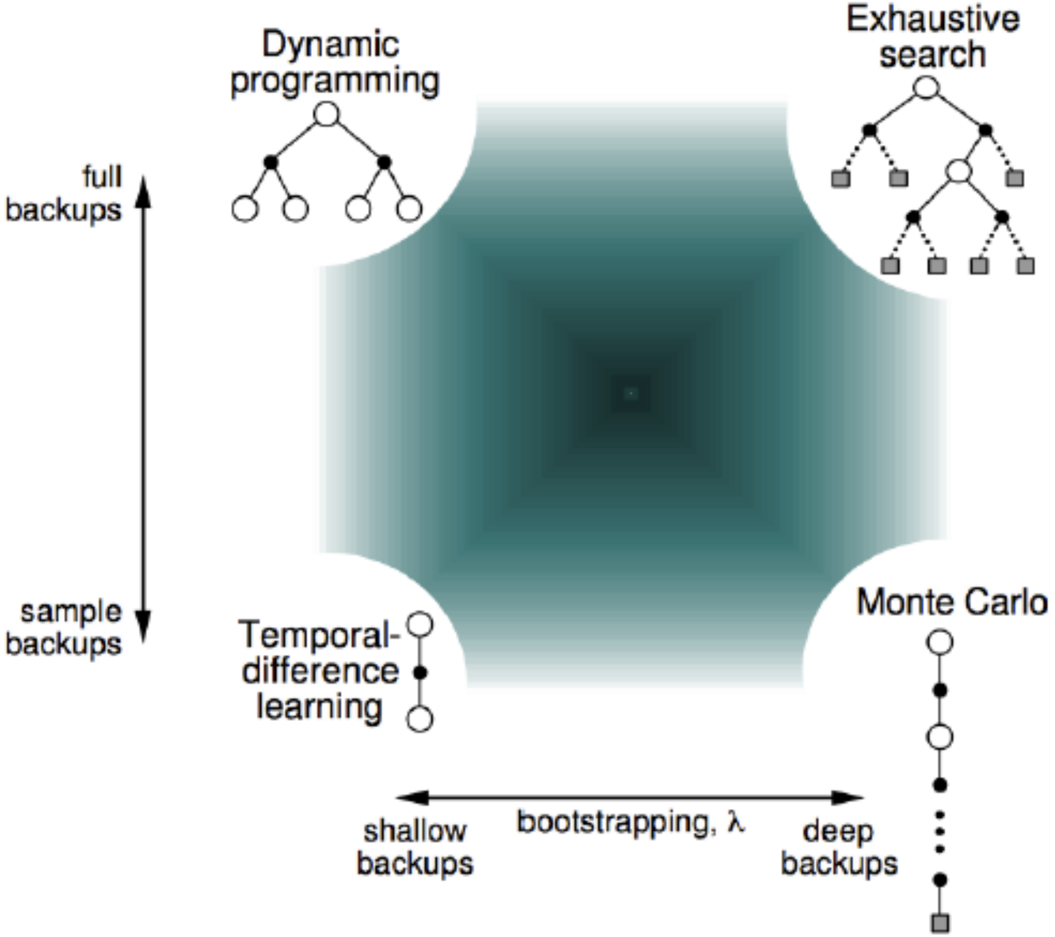
- TD exploits Markov property
    - Usually more efficient in Markov environments
- MC does not exploit Markov property
    - Usually more effective in non-Markov environments

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



TD Algorithm

For this part of the problem assume that the model shown above is not available, and we are executing the Temporal Difference (TD) algorithm to estimate the Value Function. Consider the following set of transitions:

(s0,a0) (r = 3) → (s0,a0) (r = 3) → (s2,a0) (r = -1) → (s0,a1)(r = -2)→(s2,a1)

(1) Using this data, use the TD algorithm to estimate the V values for the states s0 and s2.

Applying the TD recursion for V Values:

$$V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$$

V(S0) = 0 + 0.8(3 + 0 - 0) = 2.4

V(S0) = 2.4 + 0.8(3 + 0 – 2.4) = 2.88

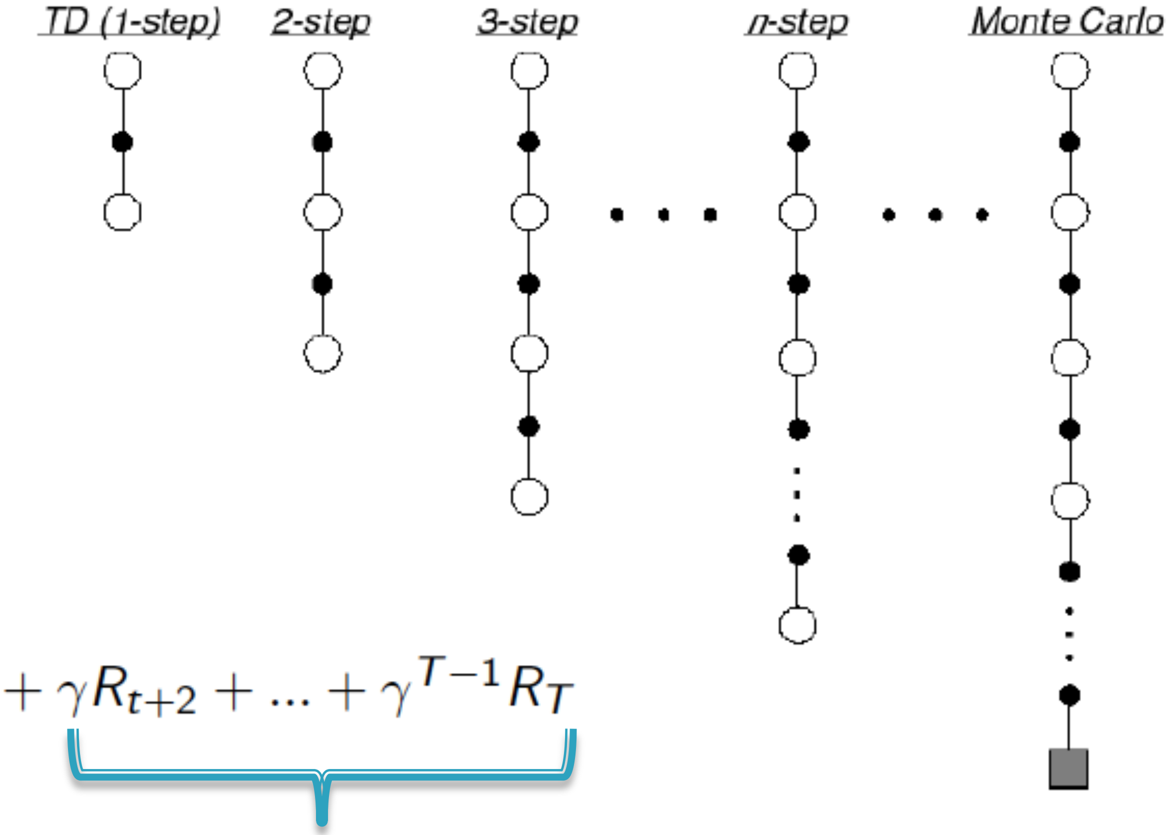
V(S2) = 0 + 0.8(-1 + 2.88 – 0) = 1.5

V(S0) = 2.88 + 0.8(-2 + 1.5 – 2.88) = 0.17

n-Step Temporal Difference: TD(n)

1–step Prediction

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



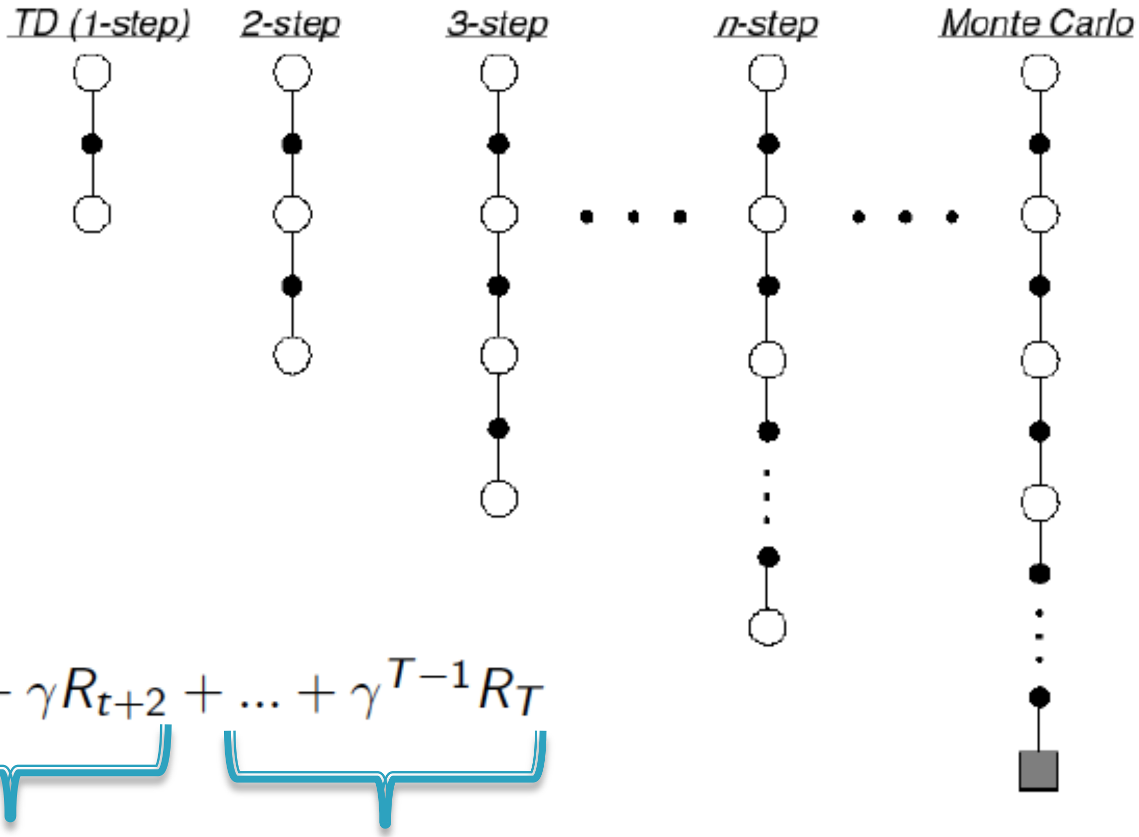
$$G_t = R_{t+1} + \underbrace{\gamma R_{t+2} + \dots + \gamma^{T-1} R_T}_{\text{Estimated Reward for rest of trajectory}} = \gamma V(S_{t+1})$$

1–step reward

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

2–step Prediction

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



$$G_t = \underbrace{R_{t+1} + \gamma R_{t+2}}_{\text{2-step reward}} + \underbrace{\ldots + \gamma^{T-1} R_T}_{\text{Estimated Reward for rest of trajectory} \; = \; \gamma^2 V(S_{t+2})}$$

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

n–step Return

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

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

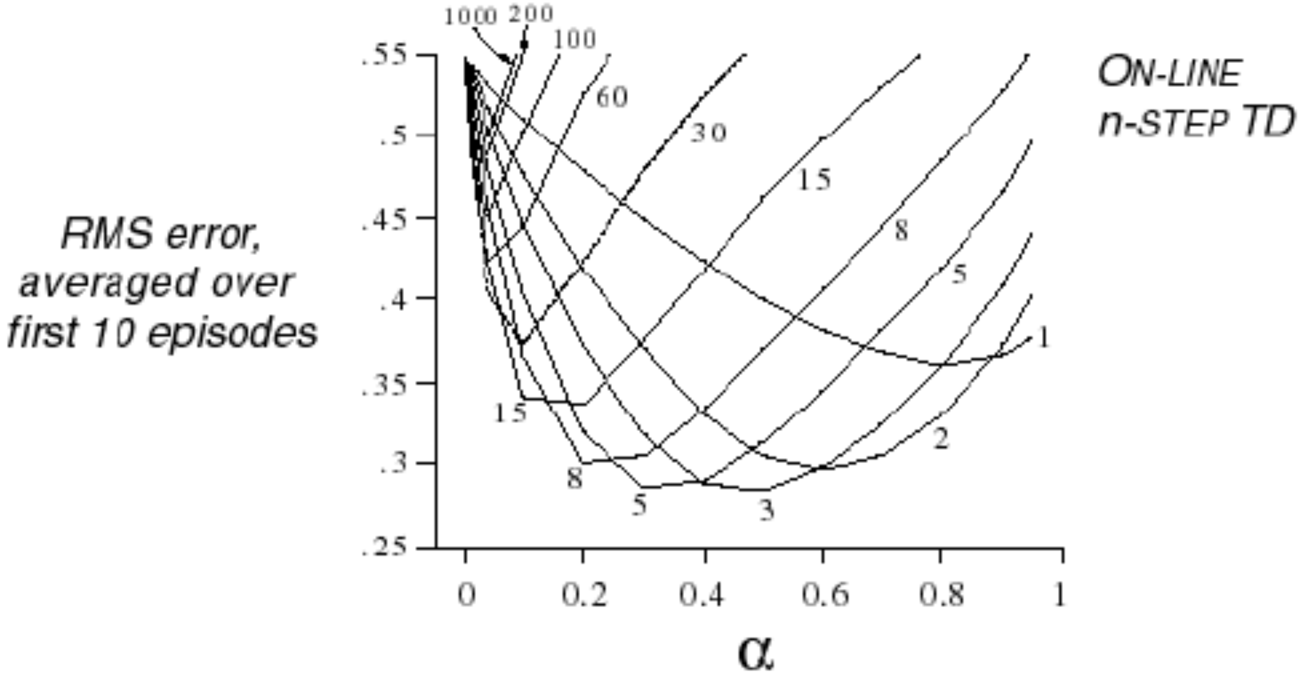
- Define the n-step return

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

- n-step temporal-difference learning

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

Large Random Walk Example



Further Reading

Sutton and Barto:

- Chapter 5: Section 5.1
- Chapter 6: Sections 6.1–6.3
- Chapter 7: Section 7.1