

Model Free Prediction Methods: Monte Carlo and Temporal Difference Algorithms

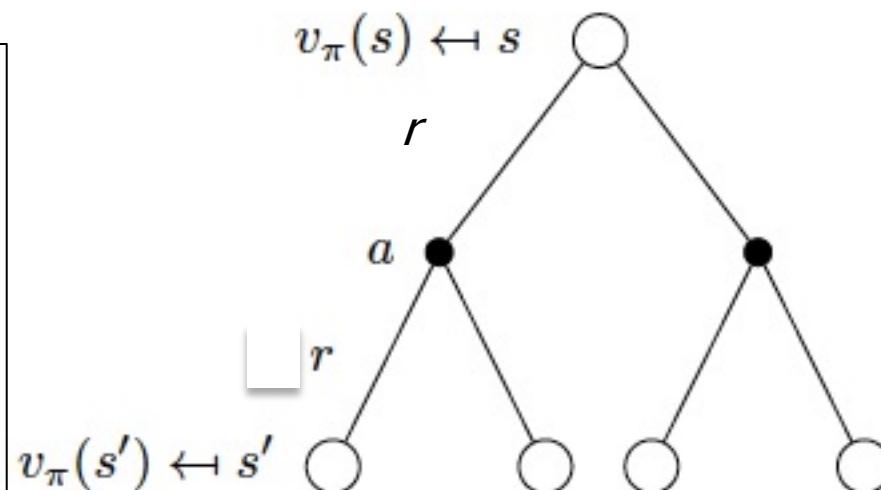
**Lecture 4
Subir Varma**

Bellman Expectation Equation for v_π

Principle of Optimality

Decompose the problem into:

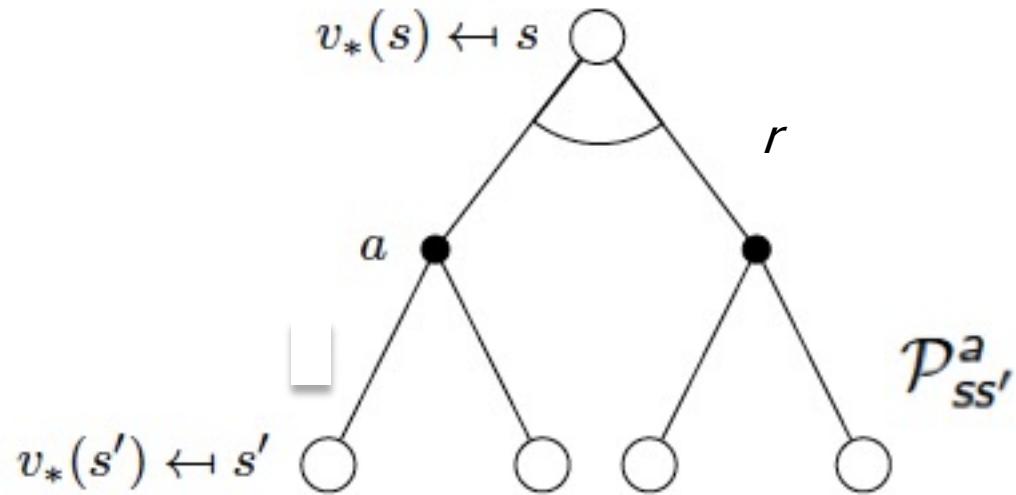
- (1) A smaller problem that is easy to solve, and
- (2) A bigger problem, that is assumed to be solved
- (3) Put the 2 parts together to solve original problem



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

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

Bellman Optimality Equation for v_*



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

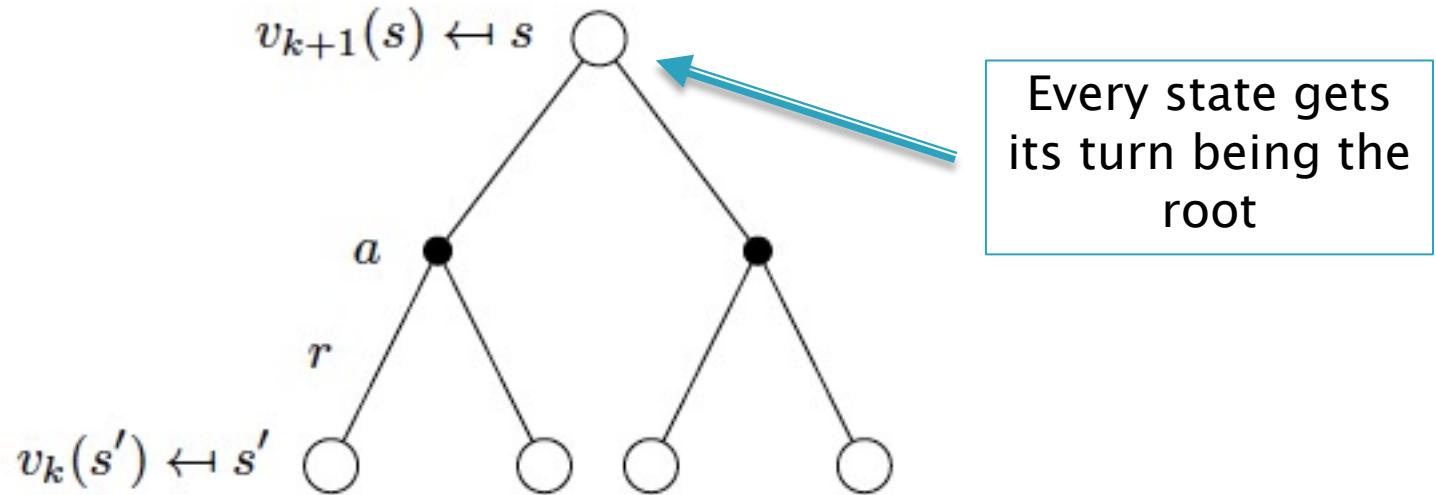
Finding the Optimal Policy

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

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

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

Iterative Policy Evaluation

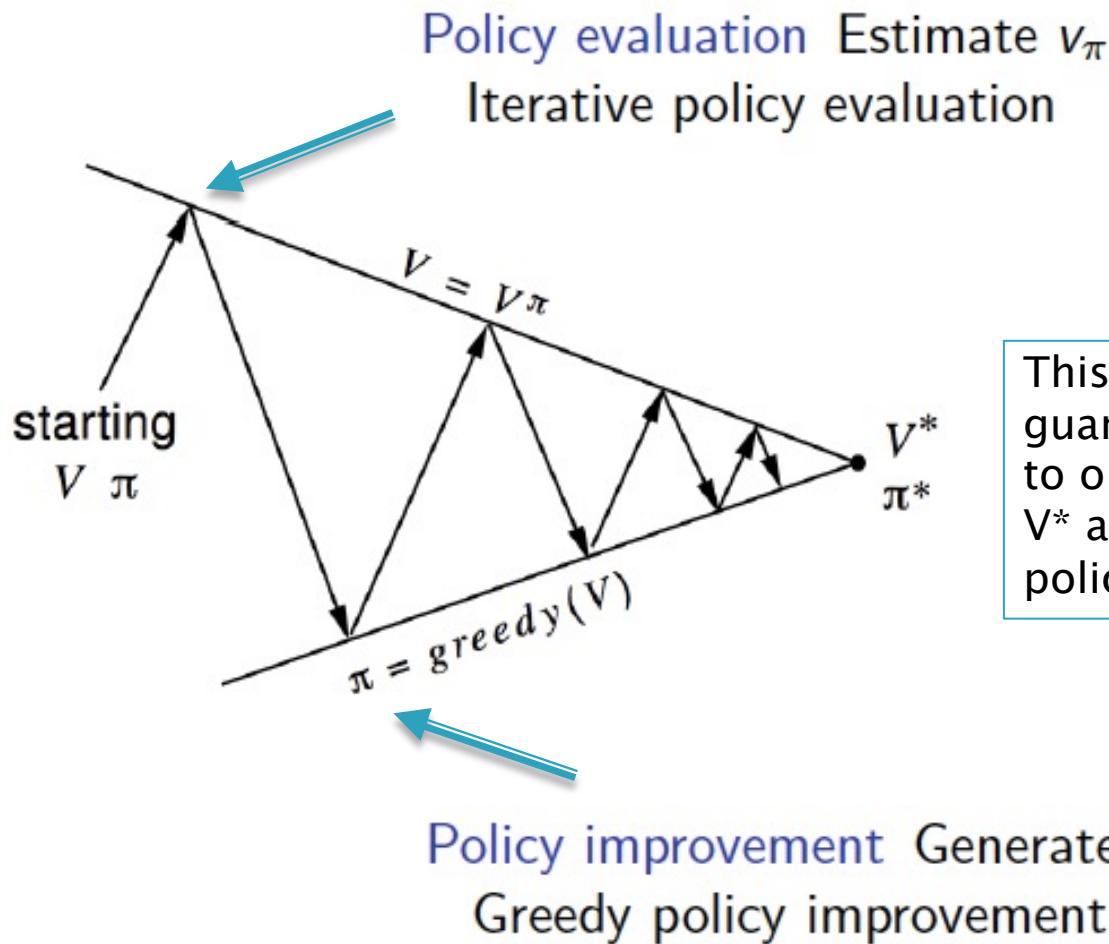


$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

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

Bellman
Expectation
Equation

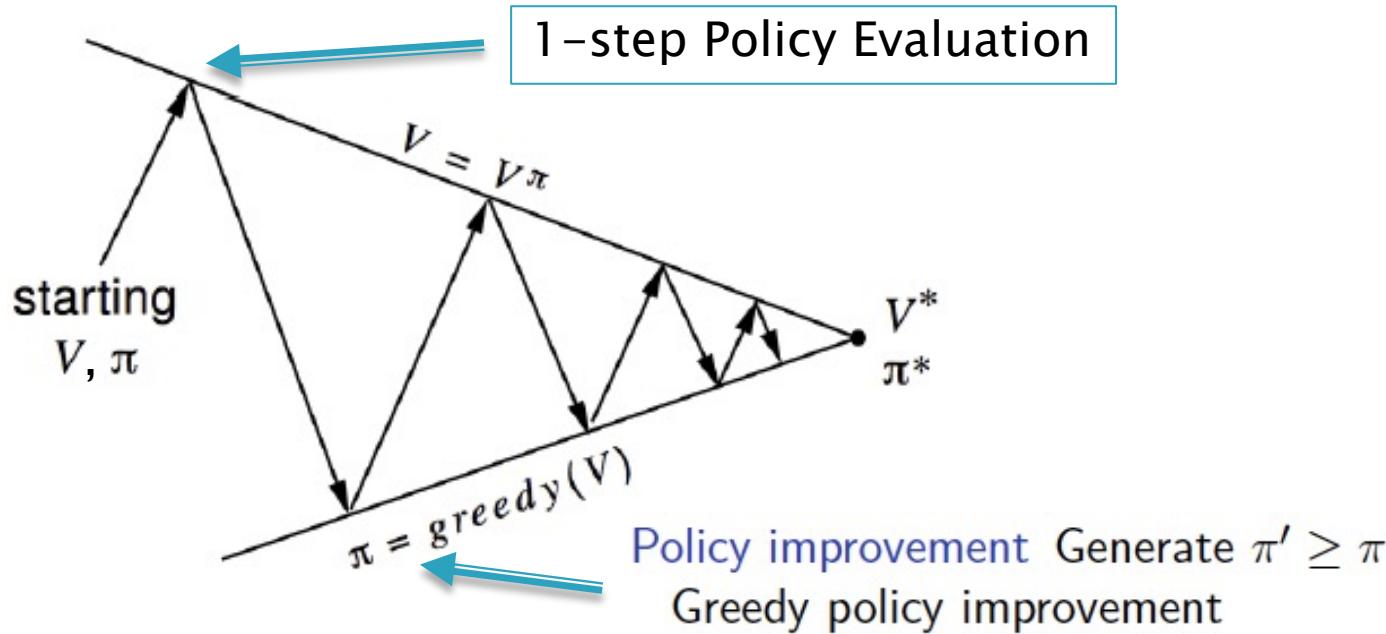
Policy Iteration – Find Best Policy



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

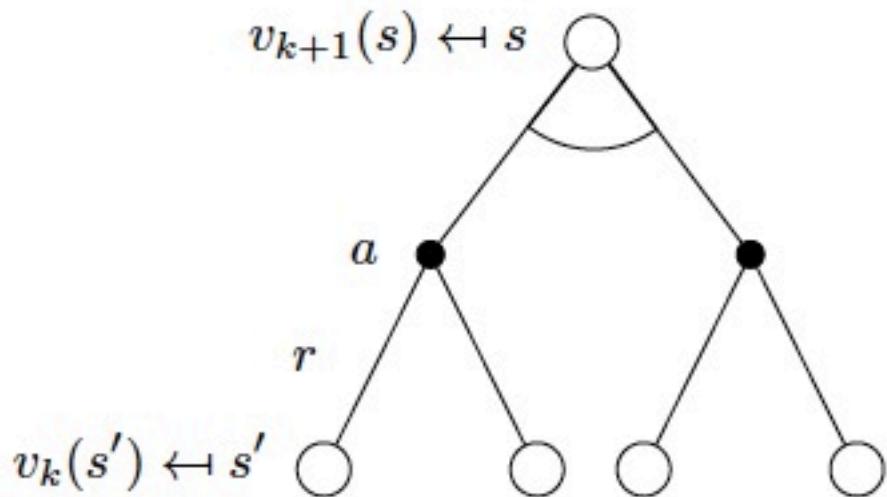
Generalized Policy Iteration

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



- 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

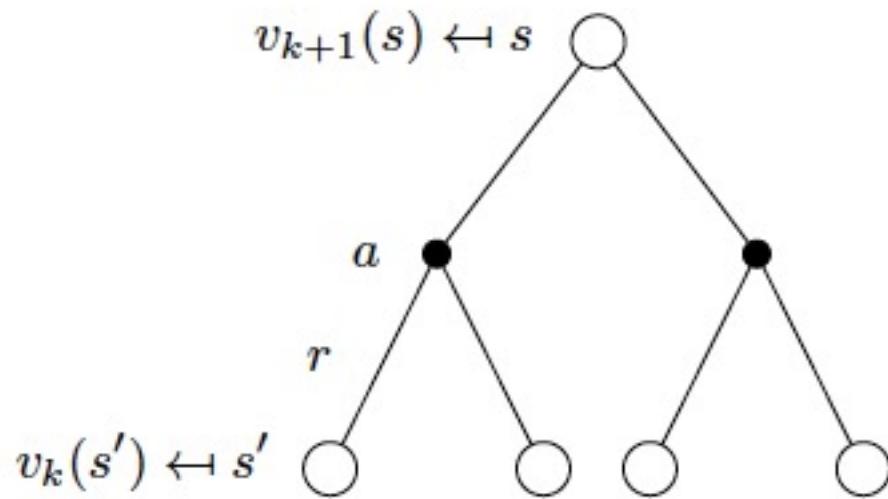


Turn the Bellman Optimality Equation into an Iterative Update

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

But...

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



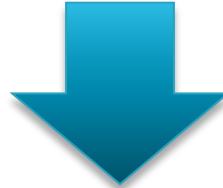
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

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

Motivation

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

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

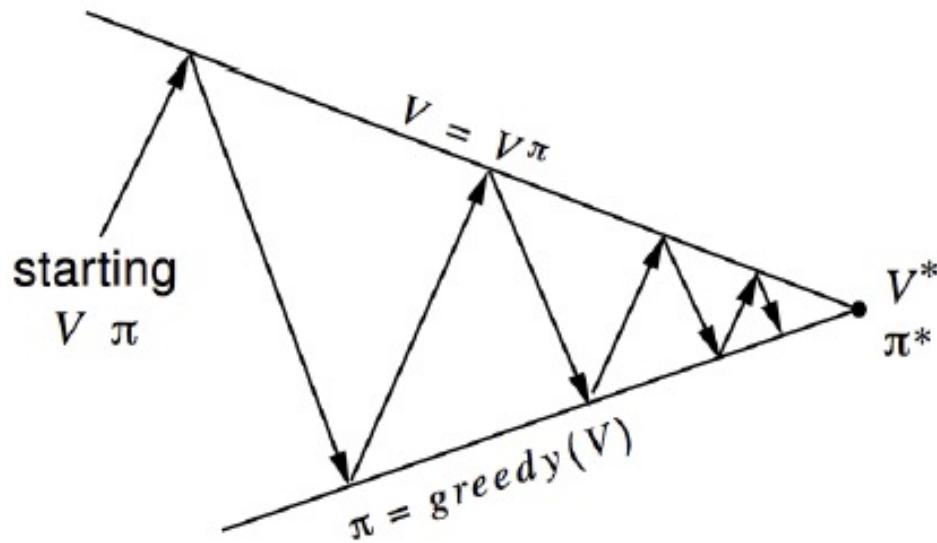


How can we find the Value Function and the Optimal Policy under these conditions



Solution: Instead of **Computing** these functions from a Model,
Learn them from Experience!

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**

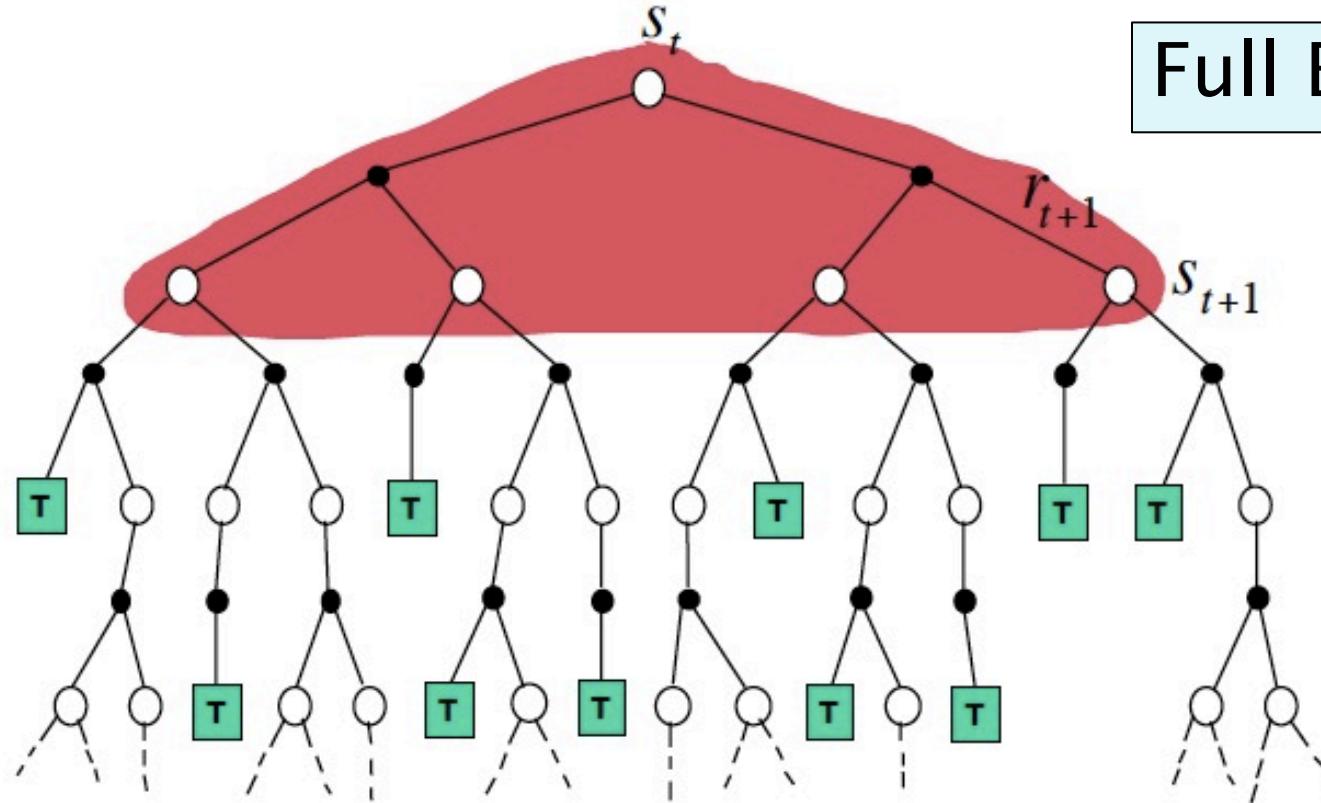
This Lecture

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

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



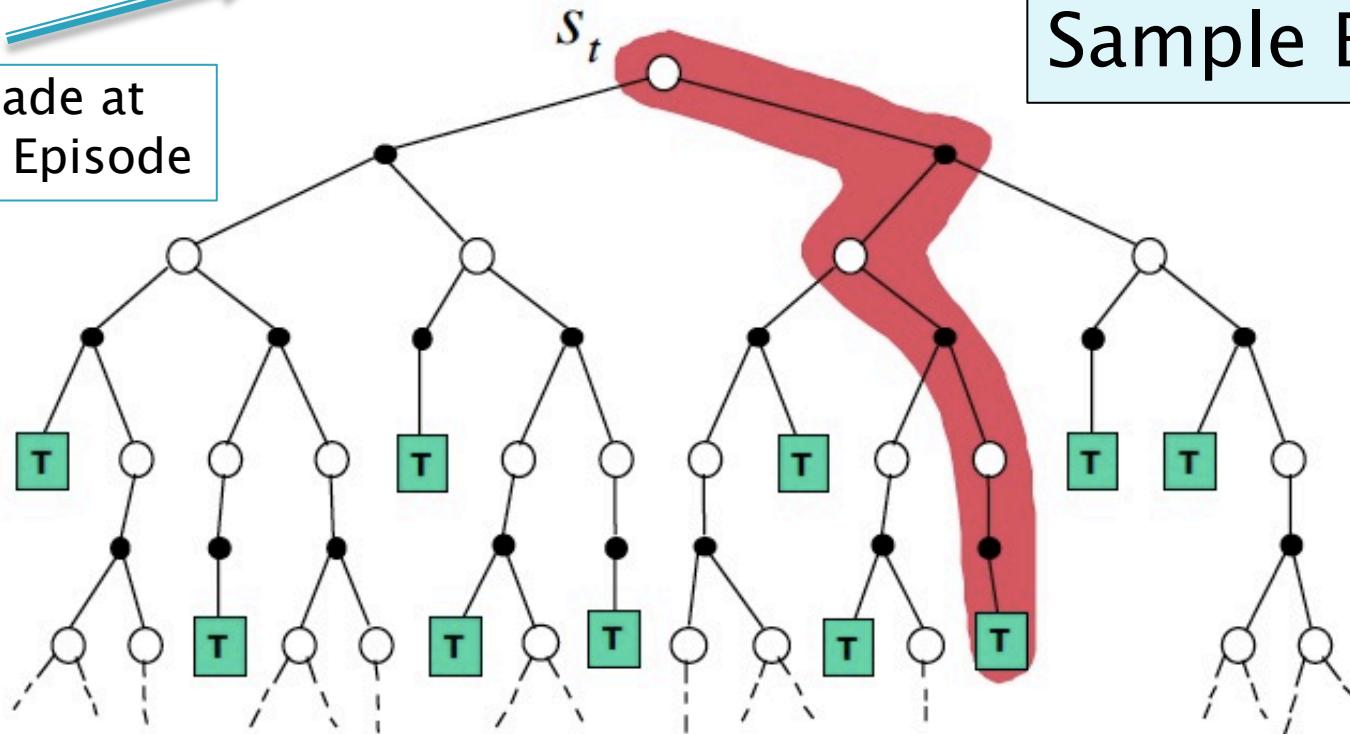
Full Backup

Monte Carlo Backup

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

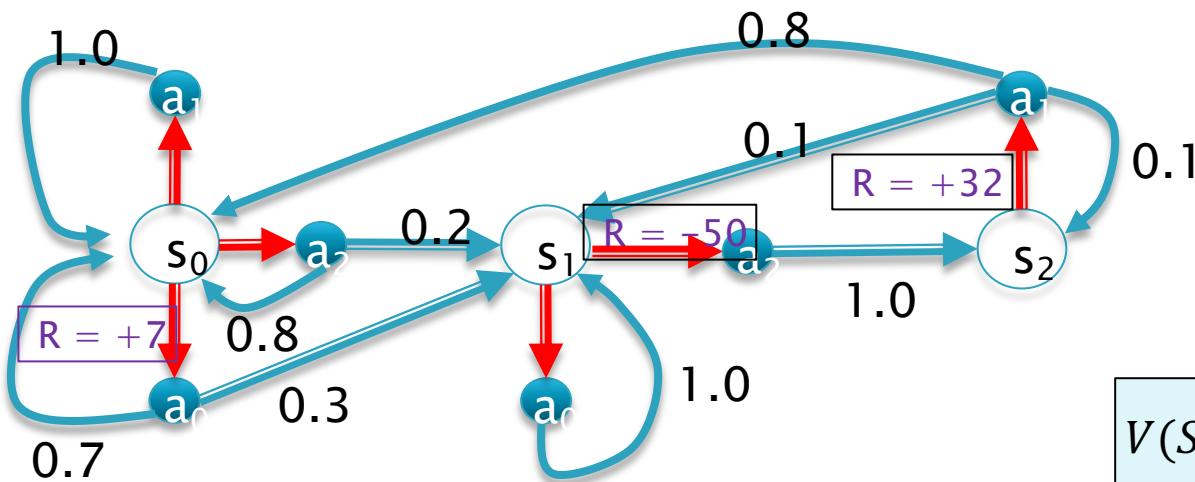
Update made at
end of an Episode

Sample Backup



Don't Need the Model Anymore!

Example: MDP Returns



Sample returns, starting from state s_0 and $\gamma = 1$,

Average return computed from 1000 episodes and 100 steps per episode

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

```
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

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:

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

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

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

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 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 π

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

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

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

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

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



New Estimate = Current Estimate + Error Term

Incremental Monte Carlo Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \dots, S_T$
- 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

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

Leads to an exponential forgetting rate

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

Sometimes also written as:

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

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 π

For each state s appearing in the episode:

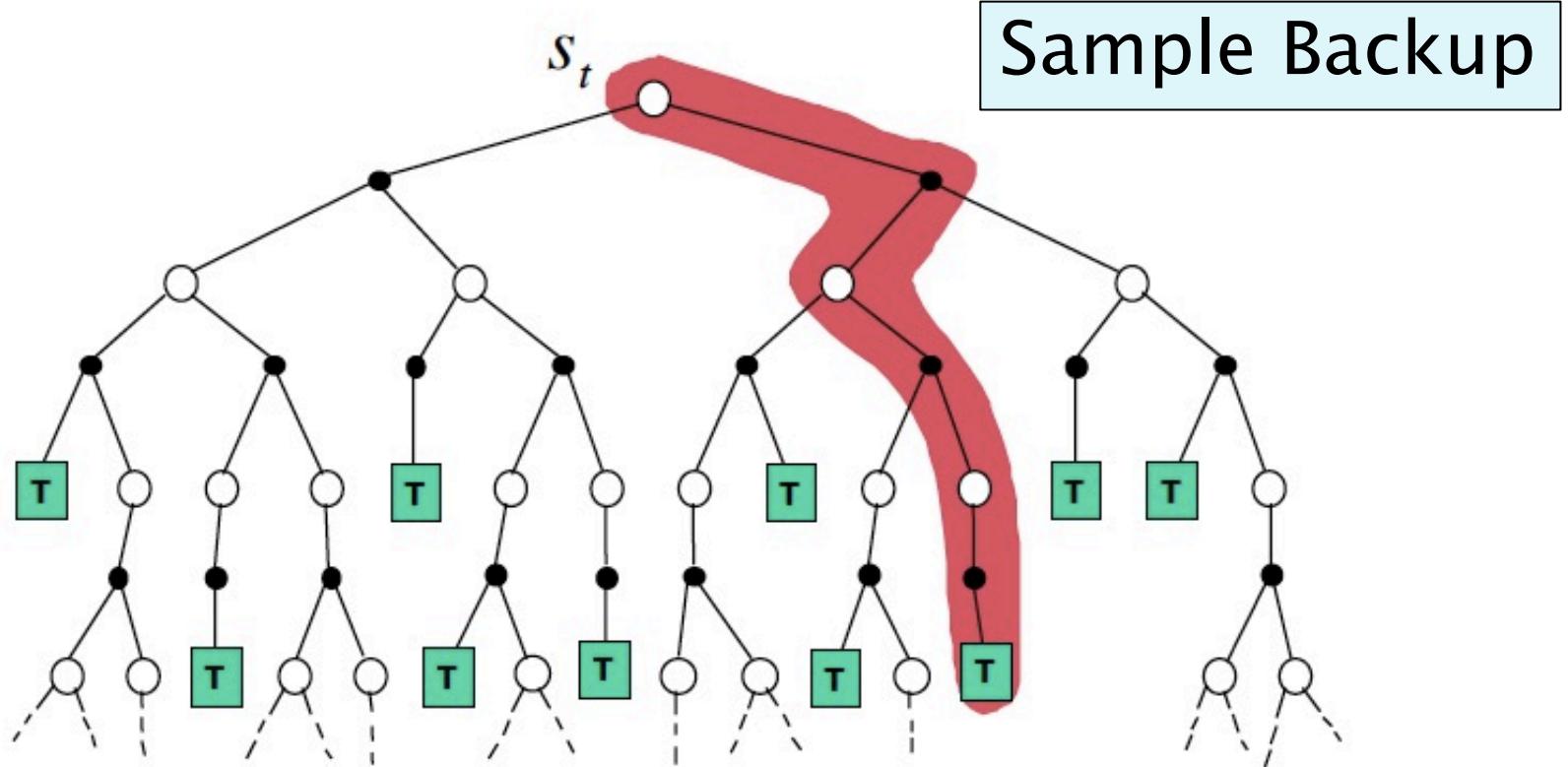
$G \leftarrow$ the return that follows the first occurrence of s

Append G to $Returns(s)$

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

MC Estimate for a Single State

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



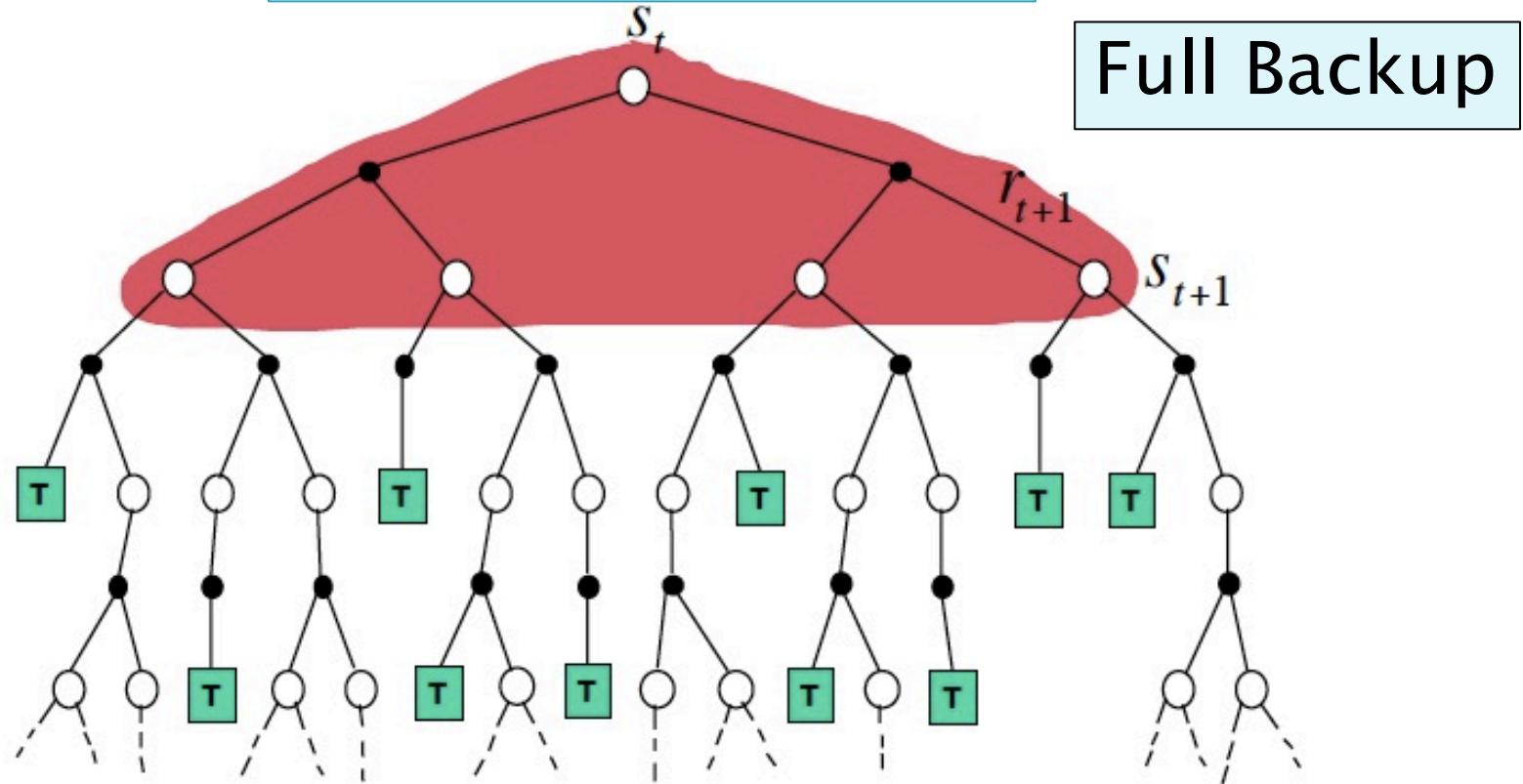
Sample Backup

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

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

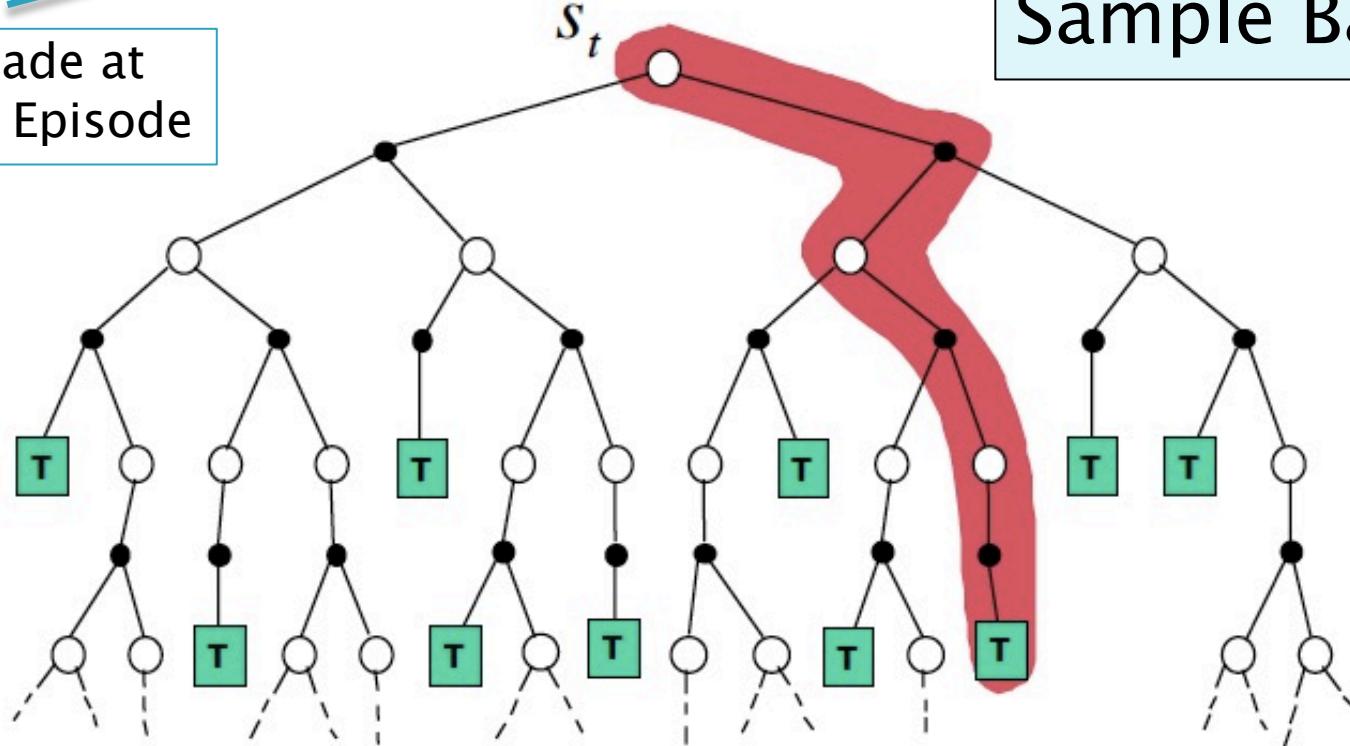


Monte Carlo Backup

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

Update made at
end of an Episode

Sample Backup



Issue with MC Learning

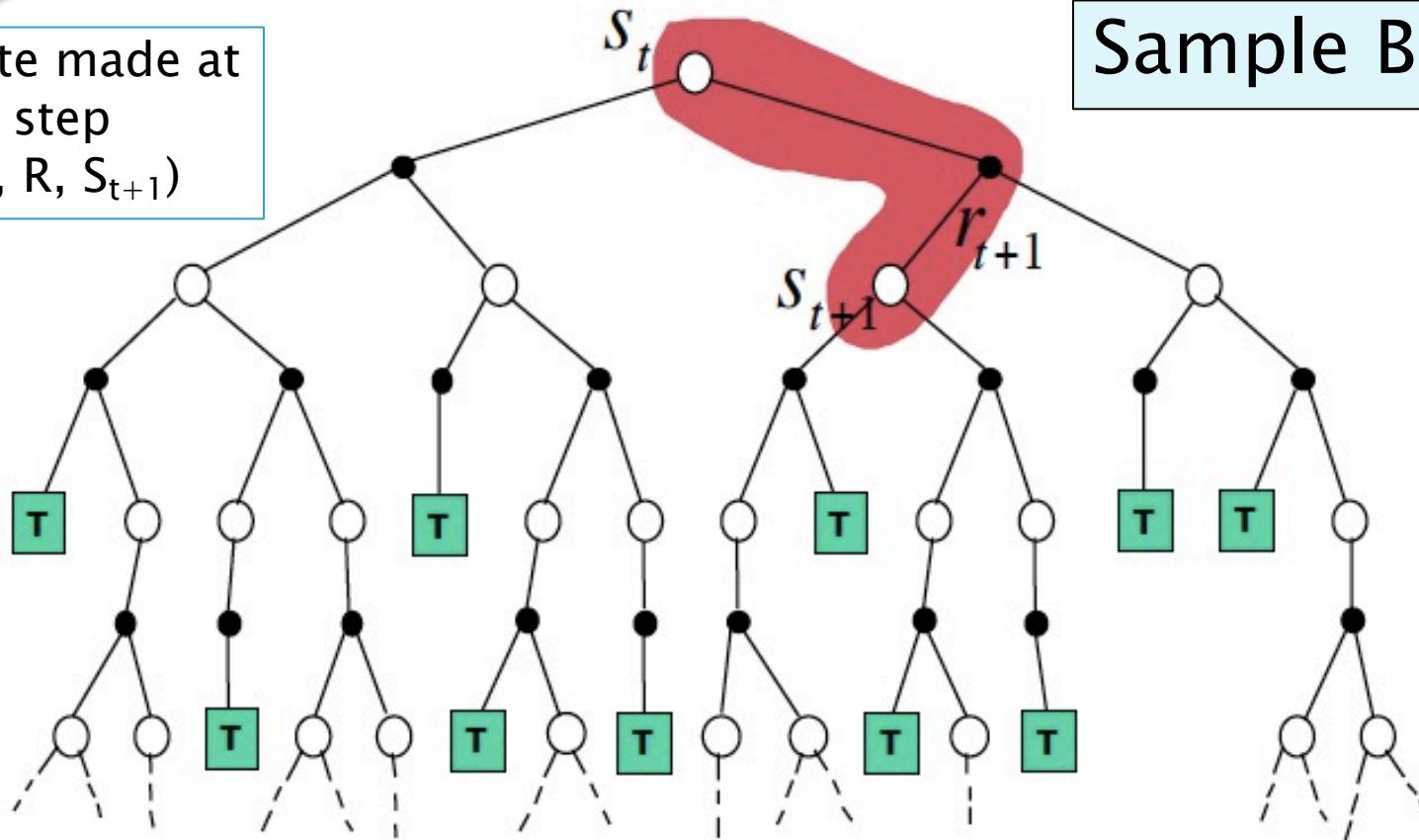
- ▶ Having to wait until end of episode to compute update
- ▶ Downsides:
 - What if something “bad” happens at end of episode
 - Some MDPs are continuous, i.e., never ending episodes

Temporal-Difference Backup

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

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

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

With TD

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

Best estimate of G_t

TD Update

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

1-step reward Estimated Reward for 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* $R_{t+1} + \gamma V(S_{t+1})$
 - $$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$
 - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
 - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

TD Algorithm

Tabular TD(0) for estimating v_π

Input: the policy π 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 π 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

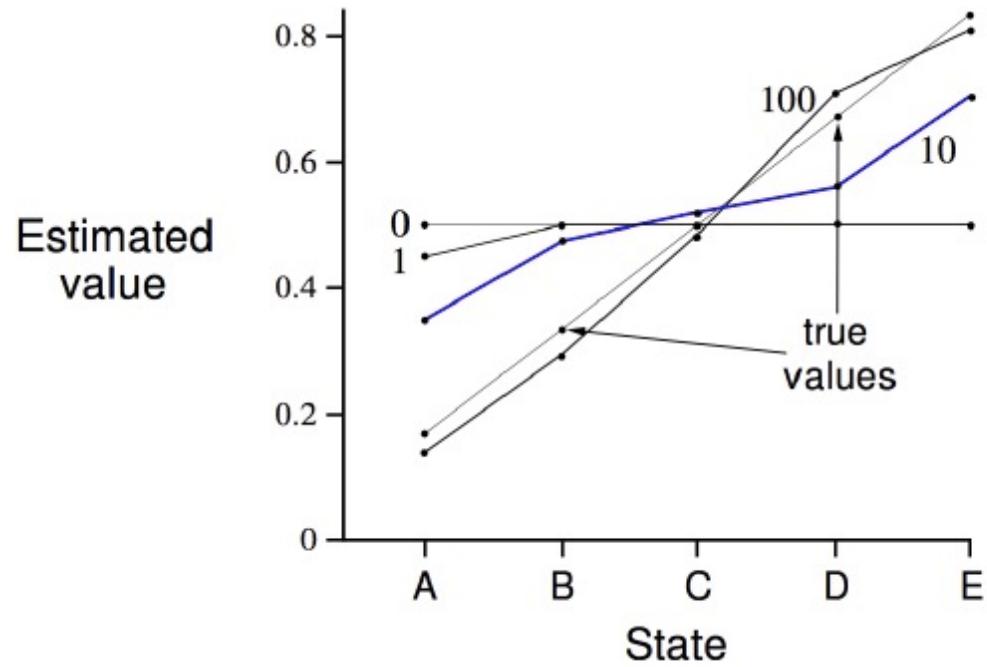
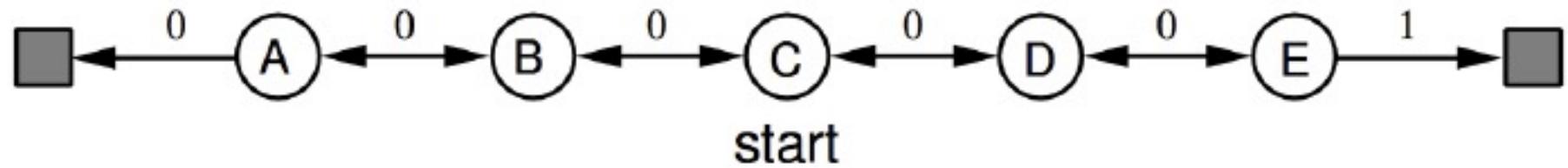
Leads to faster convergence

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

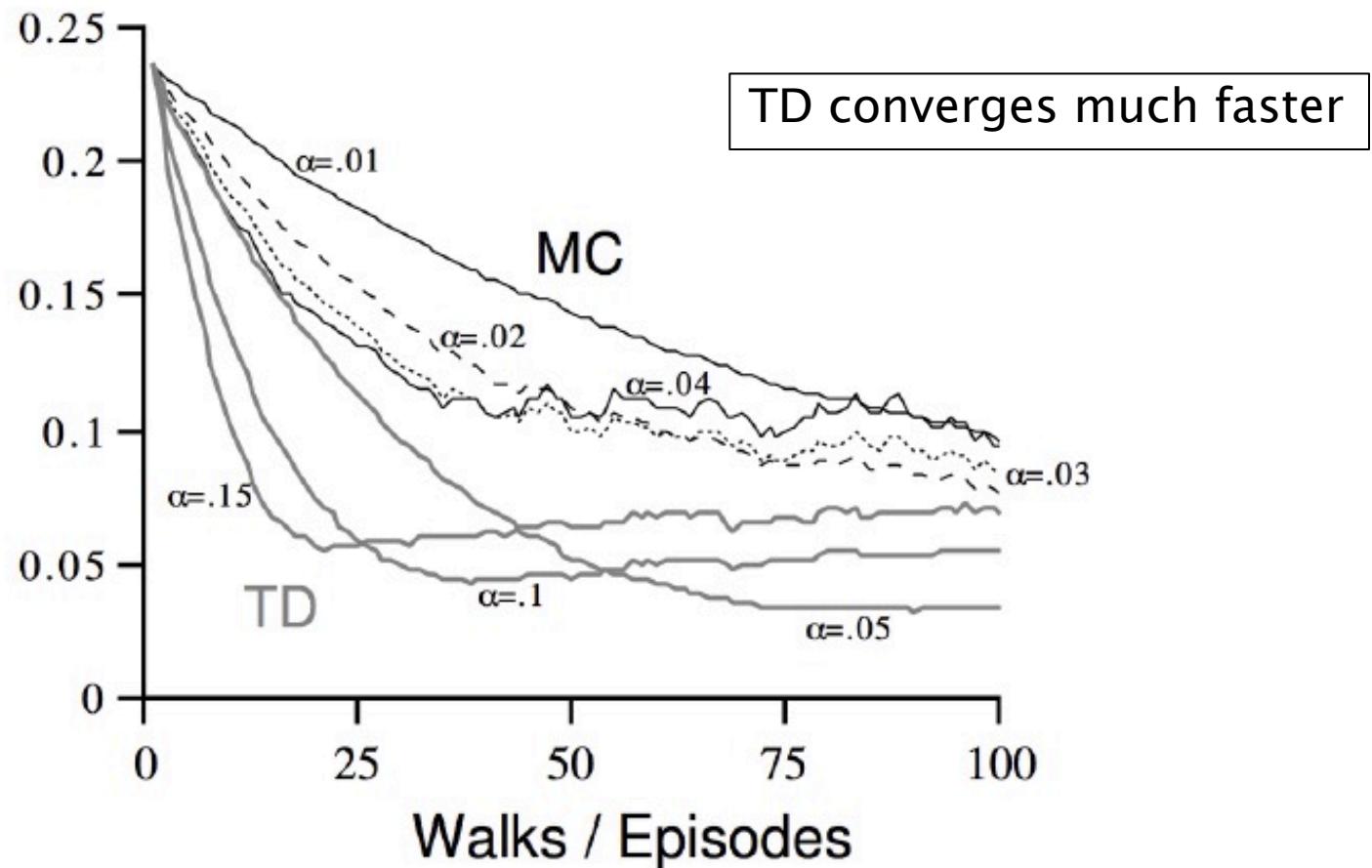
- MC has high variance, zero bias
 - Good convergence properties
 - (even with function approximation)
 - Not very sensitive to initial value
 - Very simple to understand and use
- TD has low variance, some bias
 - Usually more efficient than MC
 - TD(0) converges to $v_\pi(s)$
 - (but not always with function approximation)
 - More sensitive to initial value

Random Walk Example



Random Walk: MC vs TD

RMS error,
averaged
over states

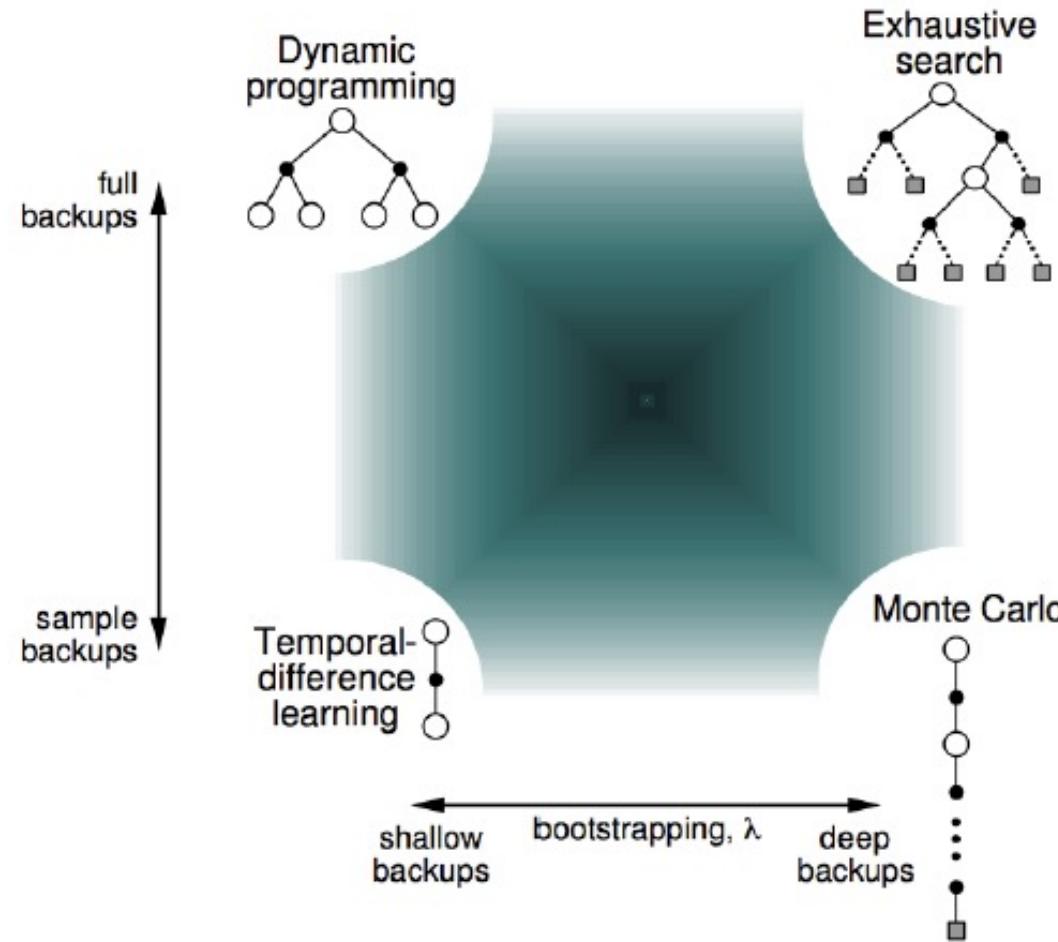


TD converges much faster

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



Further Reading

Sutton and Barto:

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