COMP 532
Machine Learning and BioInspired Optimization

Lecture 6 Reinforcement Learning

Dr. Shan Luo

Department of Computer Science

shan.luo@liverpool.ac.uk

Recap: Bellman Equation

State value function:

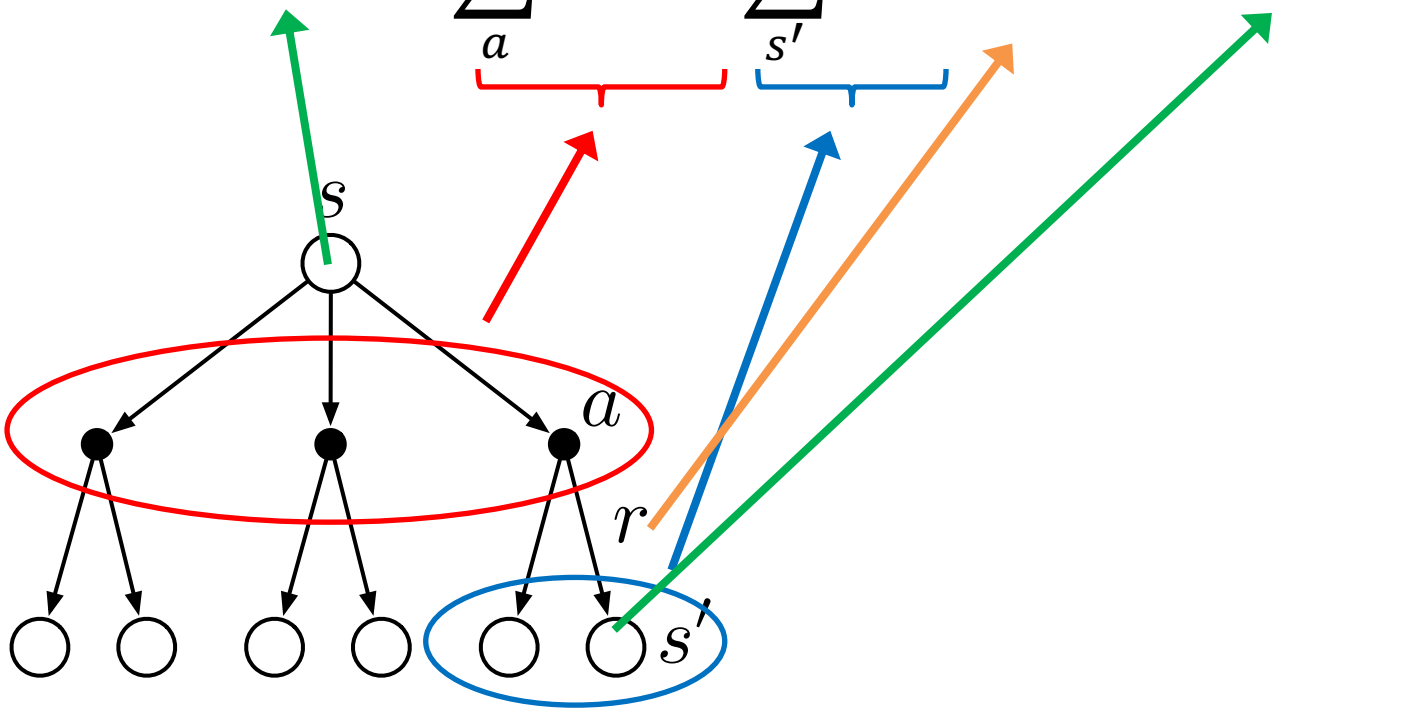
$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}$$
$$= E_{\pi}\{r_{t+1} + \gamma R_{t+1}\}$$
$$= \underbrace{\sum_{a} \pi(s,a)}_{a} \underbrace{\sum_{s'} P^{a}_{ss'}}_{s'} \underbrace{\left[R^{a}_{ss'} + \gamma V^{\pi}(s')\right]}$$

- The equation is recursive: $V(s)$ depends on $V(s')$.
- It sums over all possible future returns, weighted by their probability of occurring:
  - the action probability given by $\pi(s,a)$
  - the state transition probability given by $P^{a}_{ss'}$

Recap: Bellman Equation

State value function:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

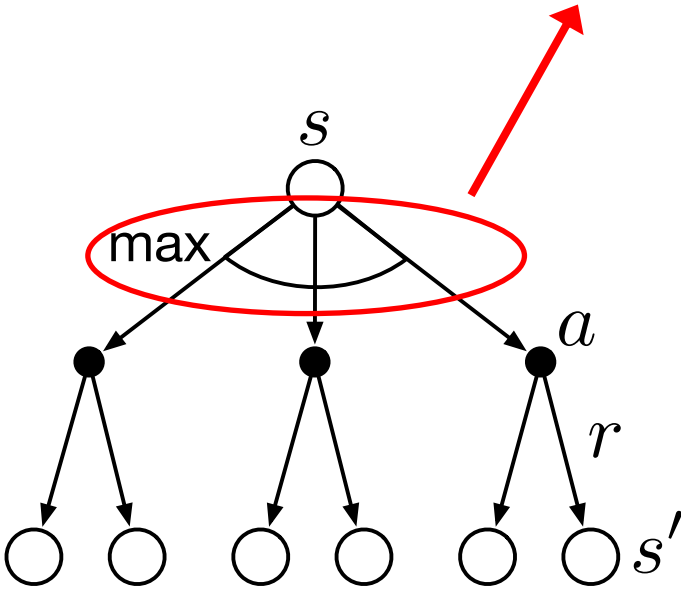


Recap: Bellman Optimality Equation

Optimal value function:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$$

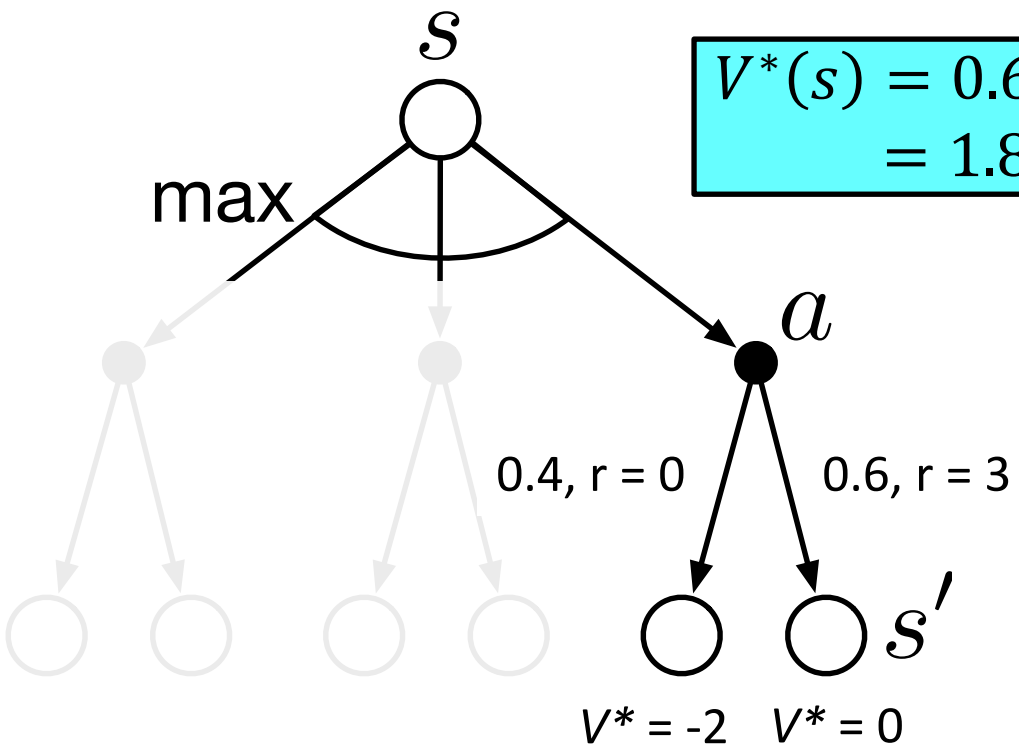
$$V^*(s) = \max_{a} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right]$$



Recap: Bellman Optimality Equation

Optimal value function:

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right]$$



$$V^*(s) = 0.6 \times [3 + 0] + 0.4 \times [0 - 2\gamma]$$
$$= 1.8 - 0.8\gamma$$

Solving the Bellman Equation

- Finding optimal policy by explicitly solving Bellman: **Dynamic Programming**

- Rarely useful in practice, it requires:
  - accurate knowledge of environment dynamics;
  - enough space and time to do the computation;
  - the Markov Property.

Solving the Bellman Optimality Equation

- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods; Chapter 4),
  - BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).

- We usually have to settle for approximations.

Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Solving the Bellman Optimality Equation

- Dynamic programming
- Monte Carlo
- Temporal Difference

Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Value Iteration

Possibility to finding optimal policy

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Bootstrapping!

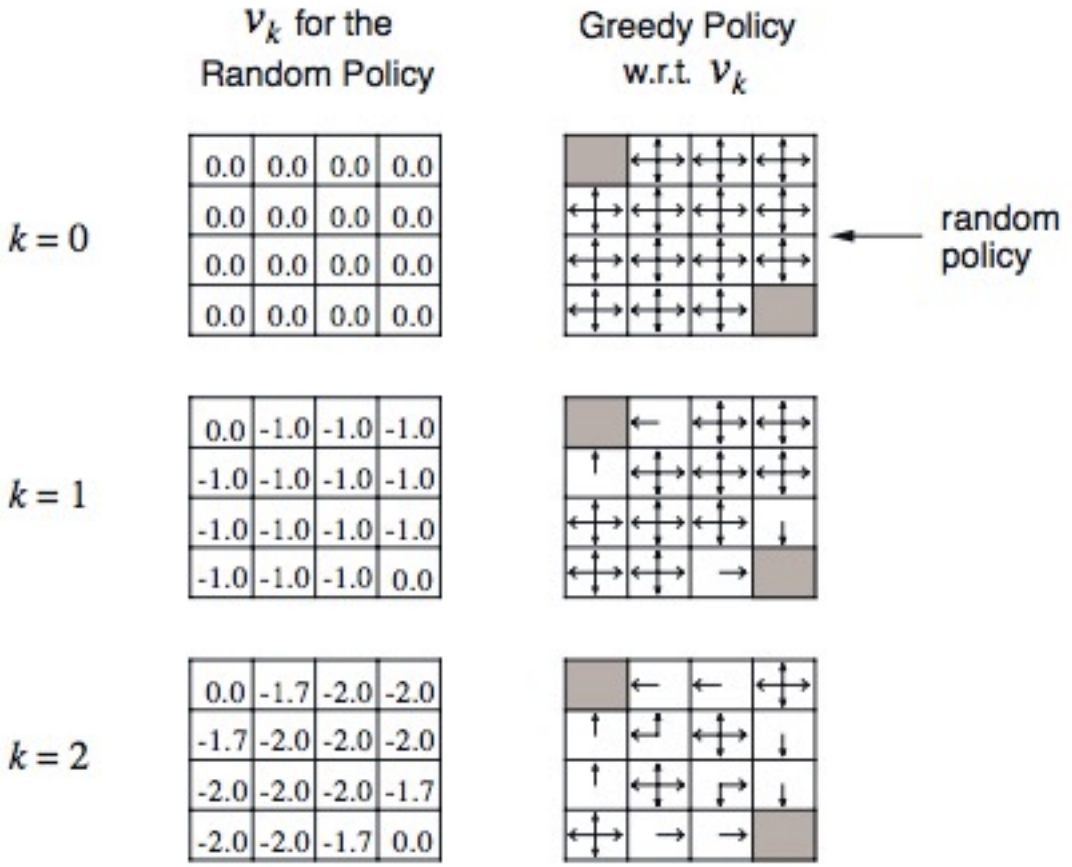
actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

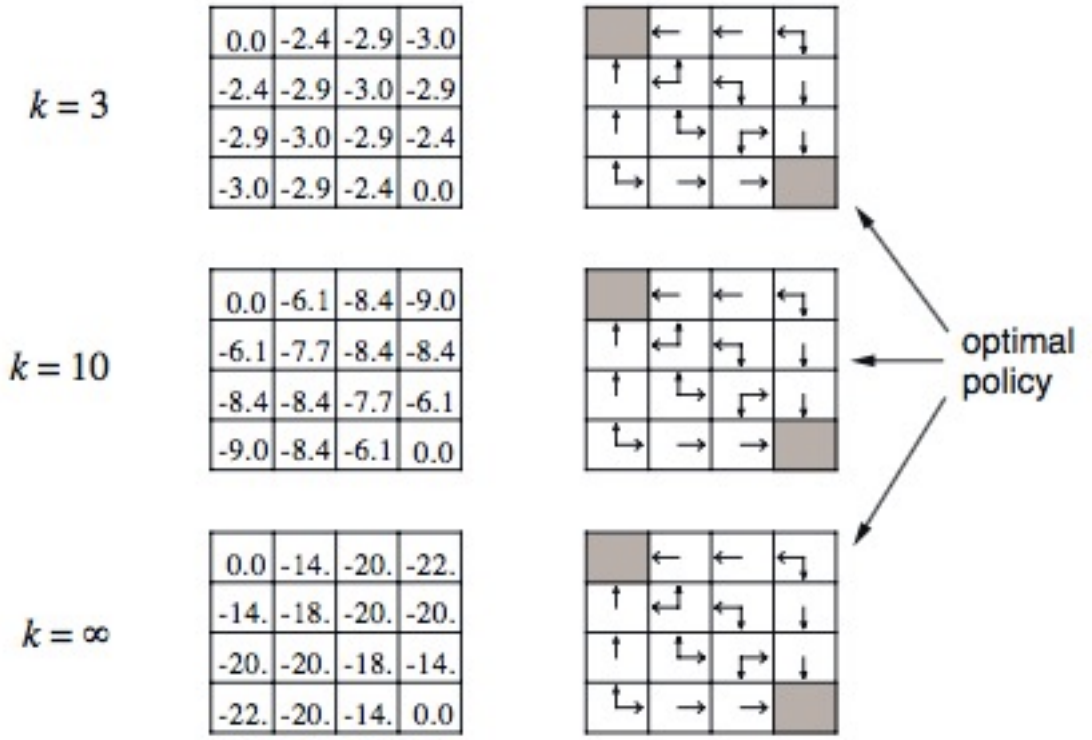
4×4 gridworld, in the book p.61

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$



4×4 gridworld, in the book p.62

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$



4×4 gridworld, in the book p.62

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

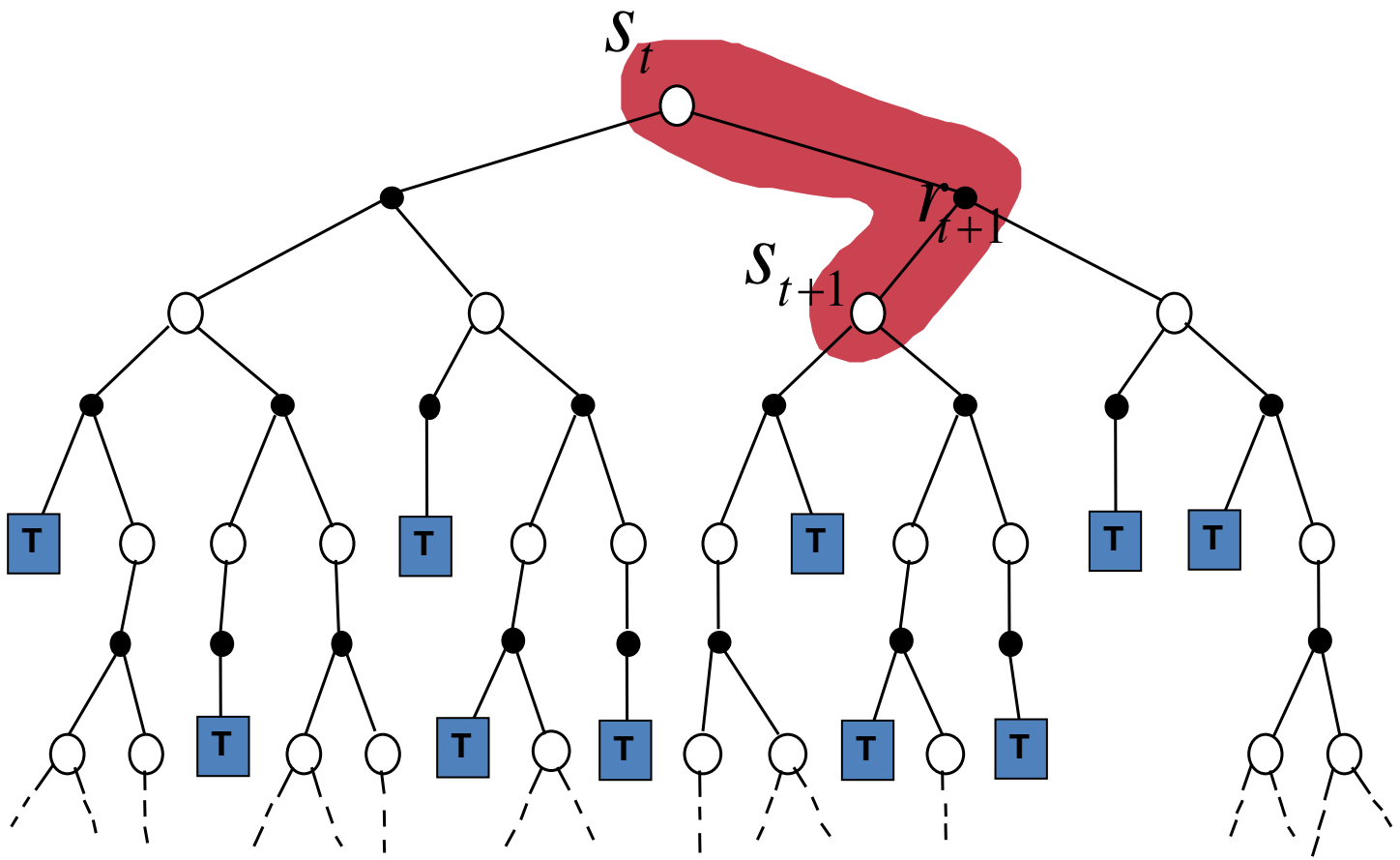
Value Iteration: Temporal Difference

No model of the environment

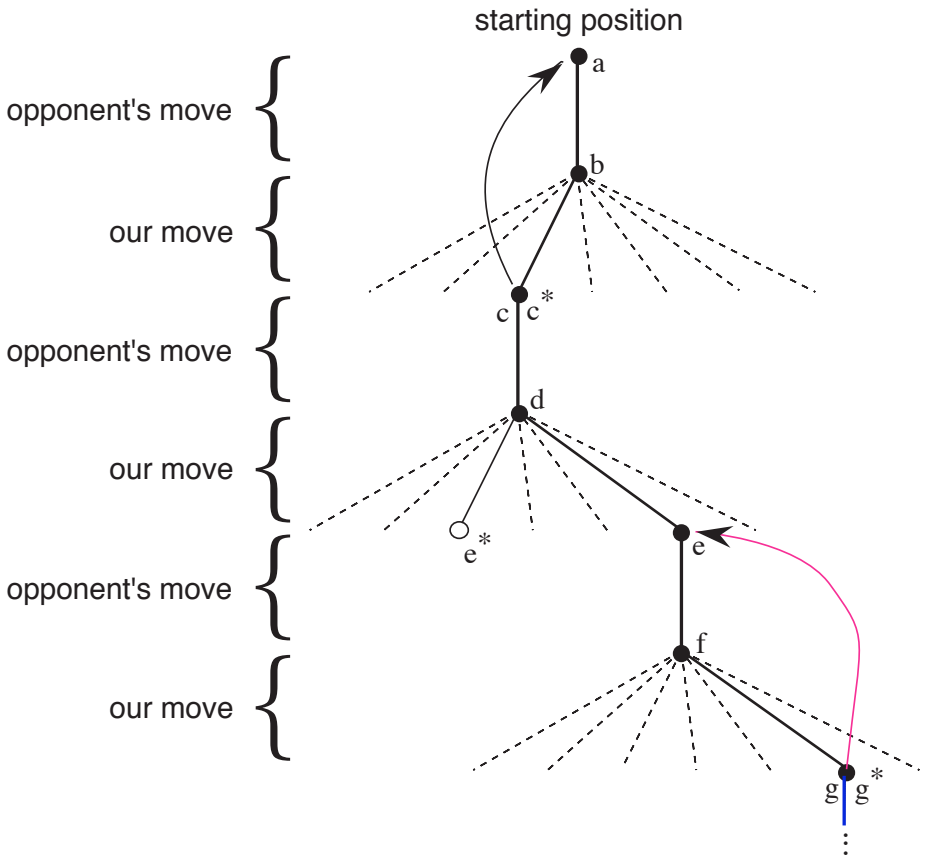
- However, the update rule requires to know the dynamics of the environment.
- Typical is to use temporal difference methods to overcome this problem, like Q-learning.
- Look at the difference between the current estimate of the value of a state and the discounted value of the next state and the reward received.

Value Iteration: Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



Remember Tic-Tac-Toe?



$$V(a) \leftarrow V(a) + \alpha[V(c) - V(a)]$$

$$V(e) \leftarrow V(e) + \alpha[V(g) - V(e)]$$

Value iteration: $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$

where r_{t+1} can be 0, and γ can be 1 !

Classification of RL methods

Model of the environment?

Bootstrap?

	YES	NO
YES	Dynamic programming	Temporal Difference (TD)
NO	_____	Monte Carlo Methods

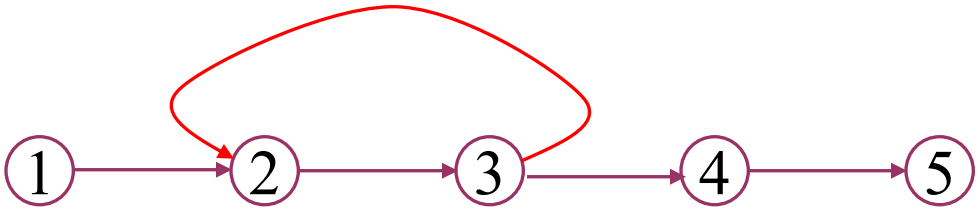
Monte Carlo Methods

- Monte Carlo methods learn from *complete* sample returns
  - Only defined for episodic tasks

- Monte Carlo methods learn directly from experience

Monte Carlo Policy Evaluation

- Goal: learn $V^{\pi}(s)$
- Given: some number of episodes under π which contain *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

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:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

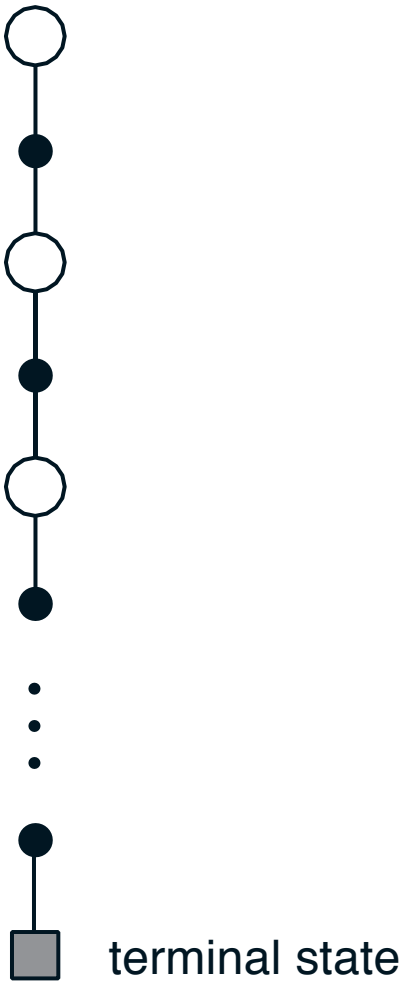
$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

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

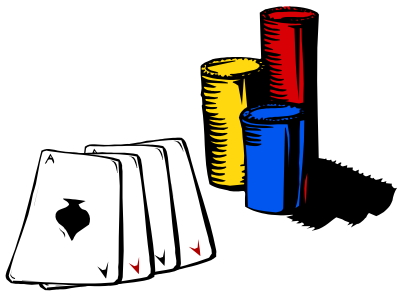
Backup diagram for Monte Carlo

- Entire episode included
- Only one choice at each state (unlike DP)
- MC does not bootstrap
- Time required to estimate one state does not depend on the total number of states



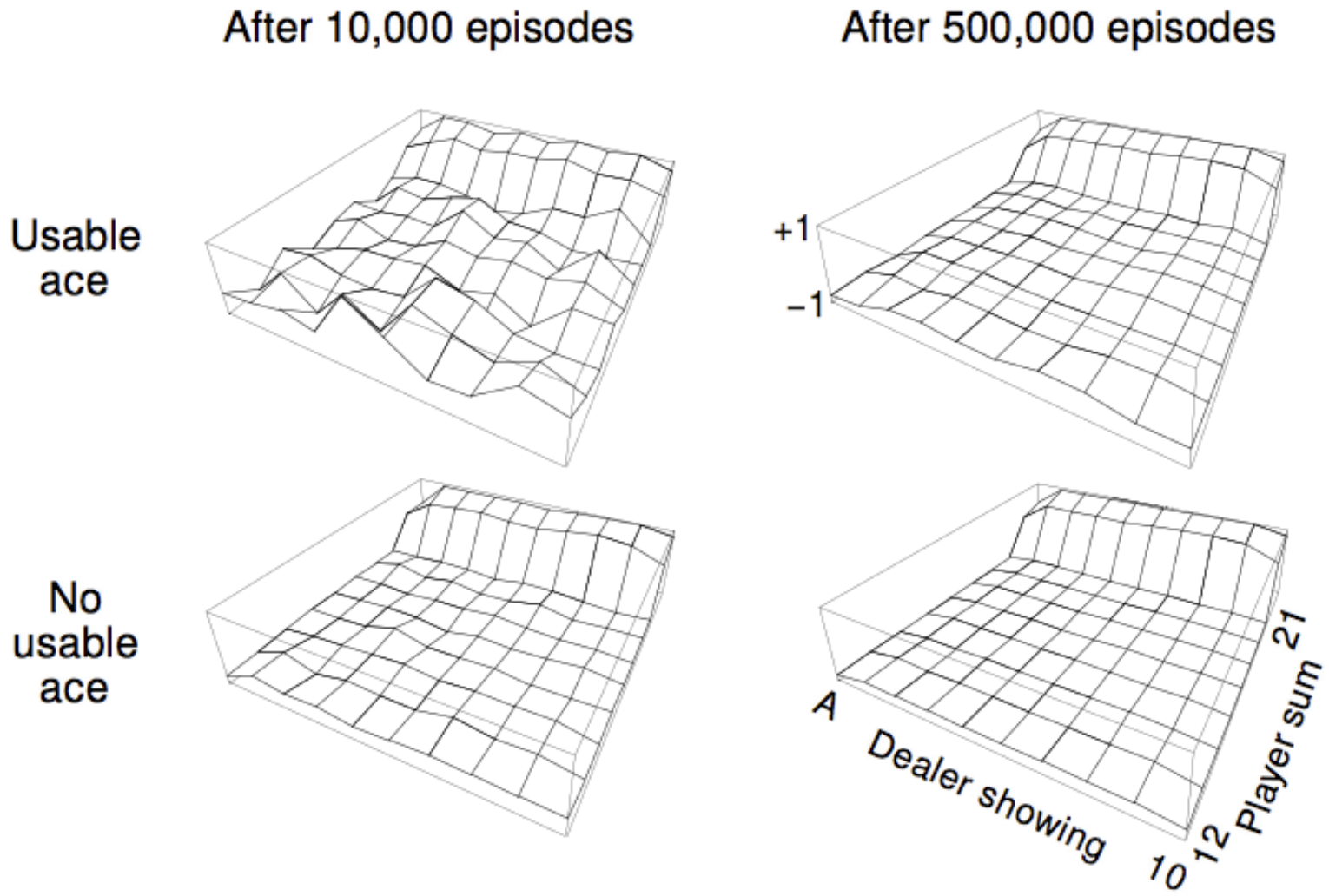
Blackjack example

- Object: Have your card sum be greater than the dealer's without exceeding 21.
- States (200 of them):
  - current sum (12-21)
  - dealer's showing card (ace-10)
  - do I have a useable ace?



- Reward: +1 for winning, 0 for a draw, -1 for losing
- Actions: stick (stop receiving cards), hit (receive another card)
- Policy: Stick if my sum is 20 or 21, else hit

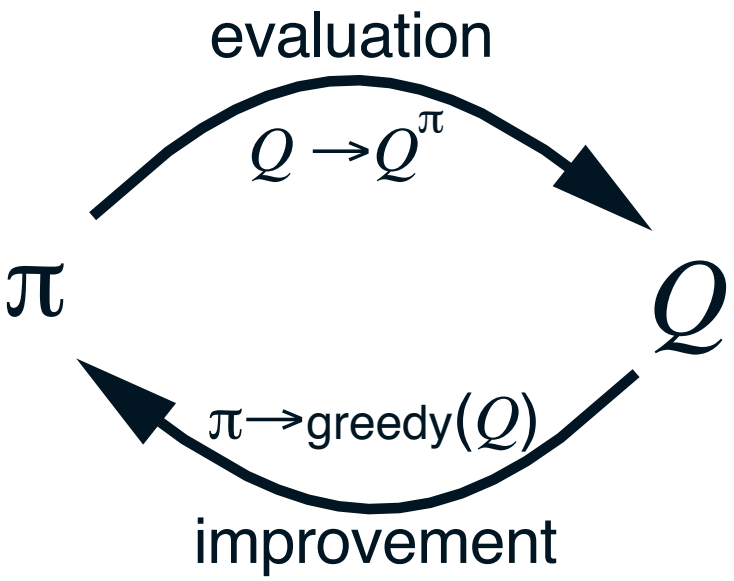
Blackjack value functions



Monte Carlo Estimation of Action Values (Q)

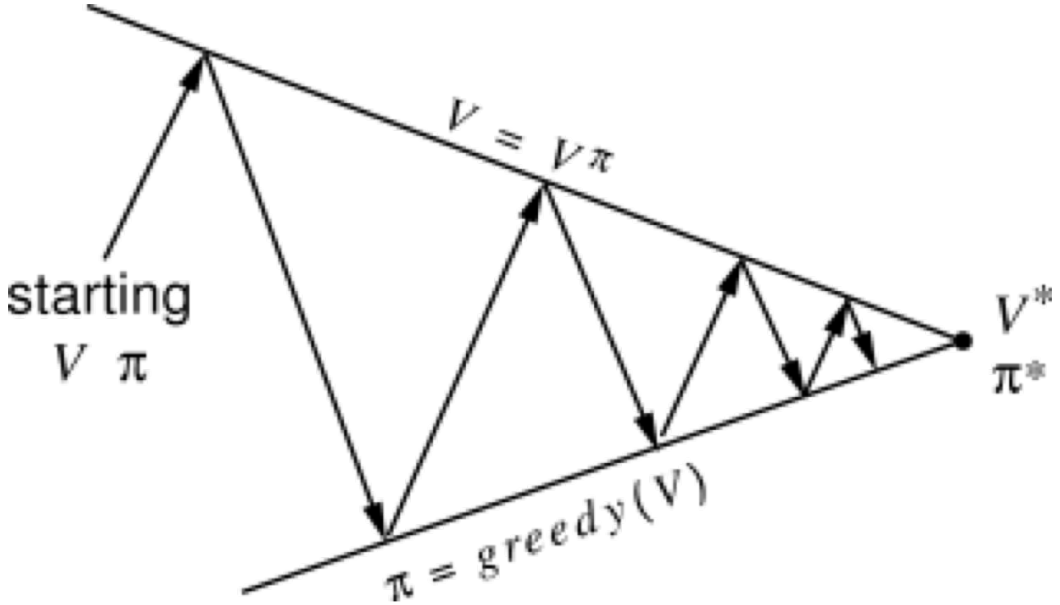
- Monte Carlo is most useful when a model is not available
  - We want to learn Q^*
- $Q^\pi(s, a)$ - average return starting from state *s* and action *a* following π
- Also converges asymptotically if every state-action pair is visited
- Exploring starts: Every state-action pair has a non-zero probability of being the starting pair

Monte Carlo Control



- MC policy iteration: Policy evaluation using MC methods followed by policy improvement
- Policy improvement step: greedy with respect to value (or action-value) function

Convergence of MC Control



Interaction between policy evaluation and improvement

iterative improvement until convergence

Convergence of MC Control

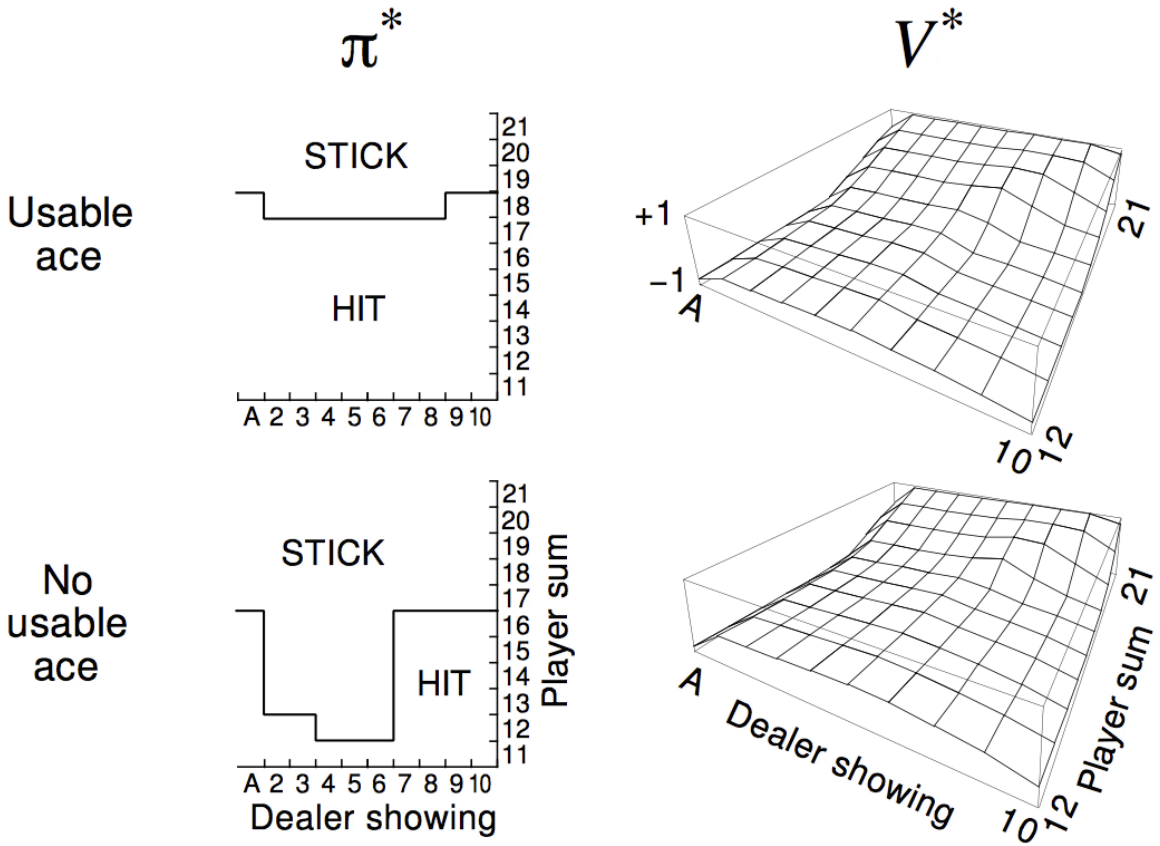
- Greedified policy meets the conditions for policy improvement:

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s, a)) \\ &= \max_a Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &\geq V^{\pi_k}(s). \end{aligned}$$

- And thus must be $\geq \pi_k$ by the policy improvement theorem
- This assumes exploring starts and infinite number of episodes for MC policy evaluation

Blackjack example continued

- Exploring starts
- Initial policy as described before



Monte Carlo is important in practice

- Absolutely
- When there are just a few possibilities to value, out of a large state space, Monte Carlo is a big win
- Backgammon, Go, …

Temporal Difference Learning

- Introduce Temporal Difference (TD) learning
- Focus first on policy evaluation, or prediction, methods
- Then extend to control methods

TD Prediction

Policy Evaluation (the prediction problem):

for a given policy π, compute the state-value function V^{π}

Recall: Simple every-visit Monte Carlo method:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

target: the actual return after time t

The simplest TD method, TD(0):

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

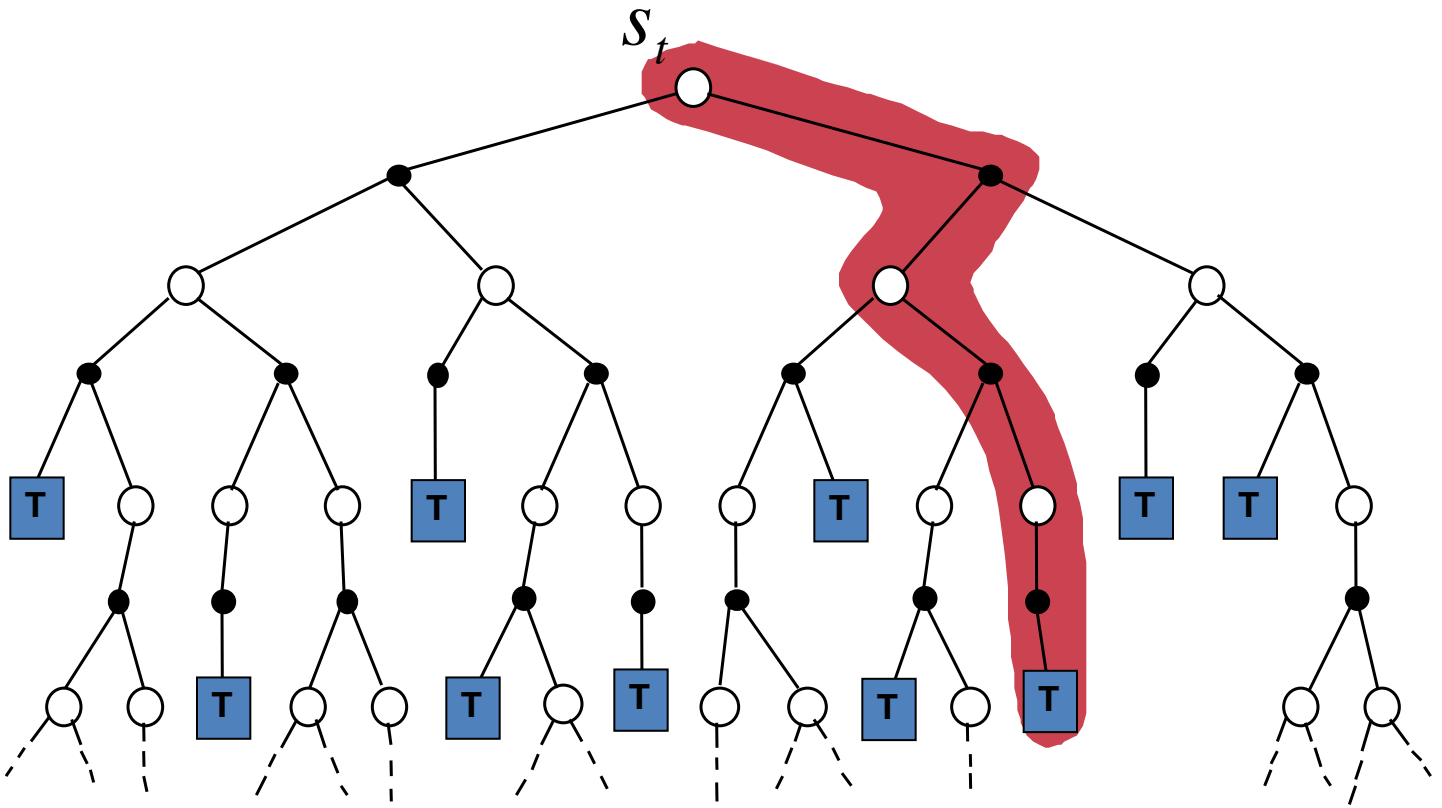
target: an estimate of the return

(TD error)

Simple Monte Carlo

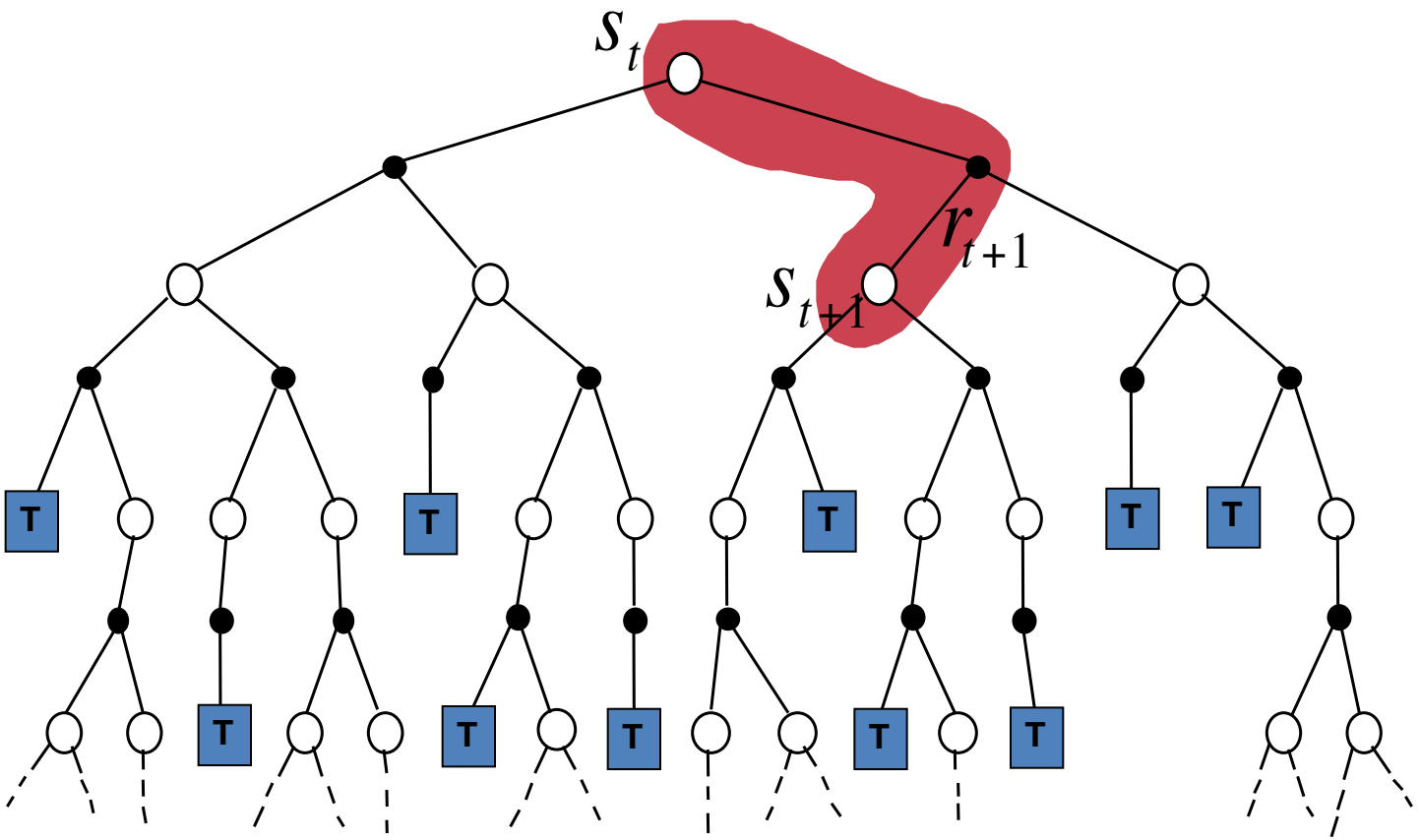
$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

where R_t is the actual return following state s_t.



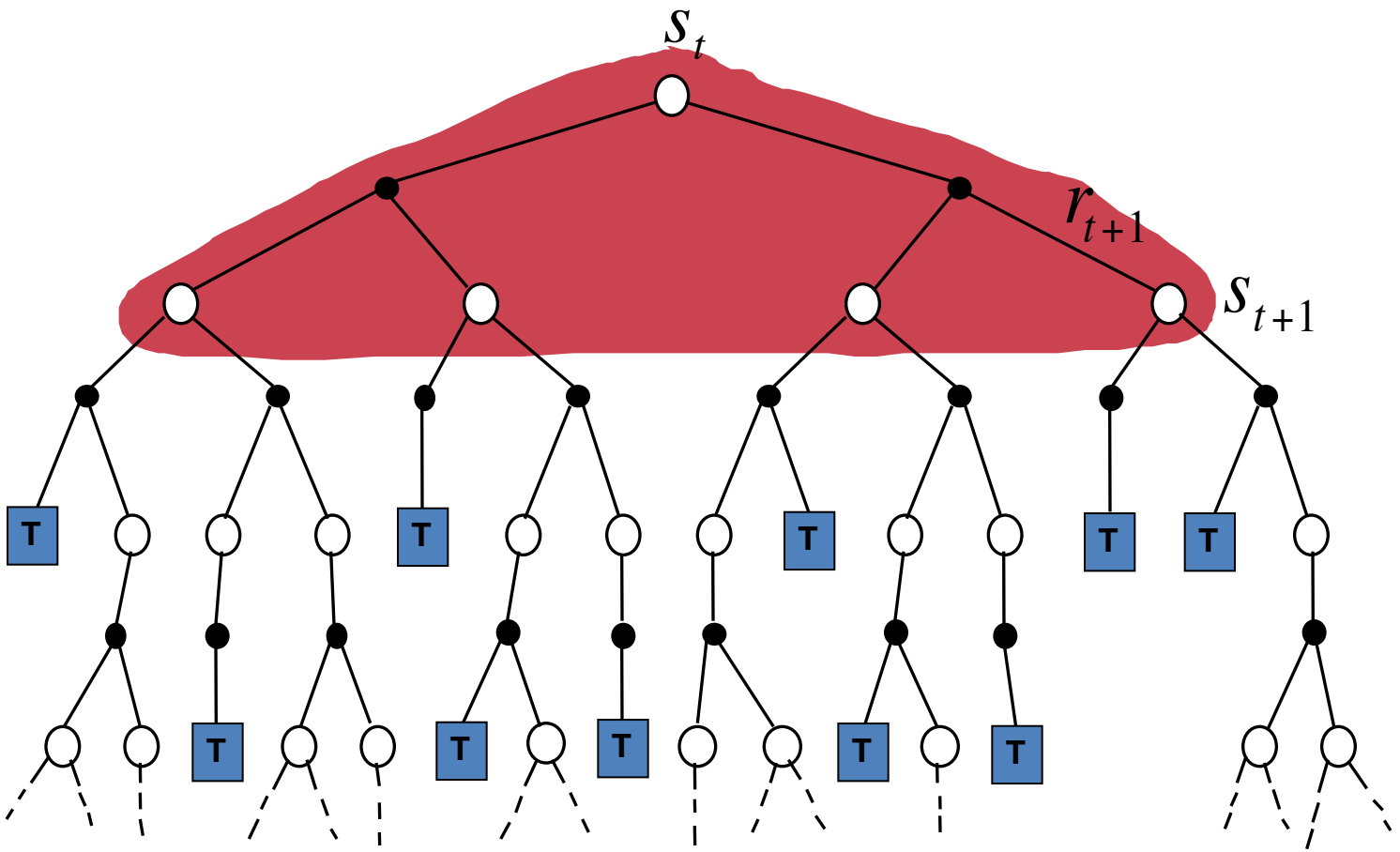
Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



Dynamic Programming

$$V(s_t) \leftarrow E_\pi\{r_{t+1} + \gamma V(s_t)\}$$



TD methods bootstrap and sample

- Bootstrapping: update estimates on the basis of other estimates
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps

- Sampling: update does not involve an *expected value*
  - MC samples
  - DP does not sample
  - TD samples

Advantages of TD Learning

- TD methods do not require a model of the environment, only experience

- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences