Reinforcement Learning

- Learn via experiences!
- Examples :
- **Game playing** - determining the best move to make in a game often depends on number of possible states that can exist in it. RL cuts out the need to manually specify rules, agents learn simply by playing the game.
- **Control problems** - such as elevator scheduling. RL agents can be left to learn in a simulated environment and eventually they will come up with good controlling policies.

Elements of Reinforcement Learning

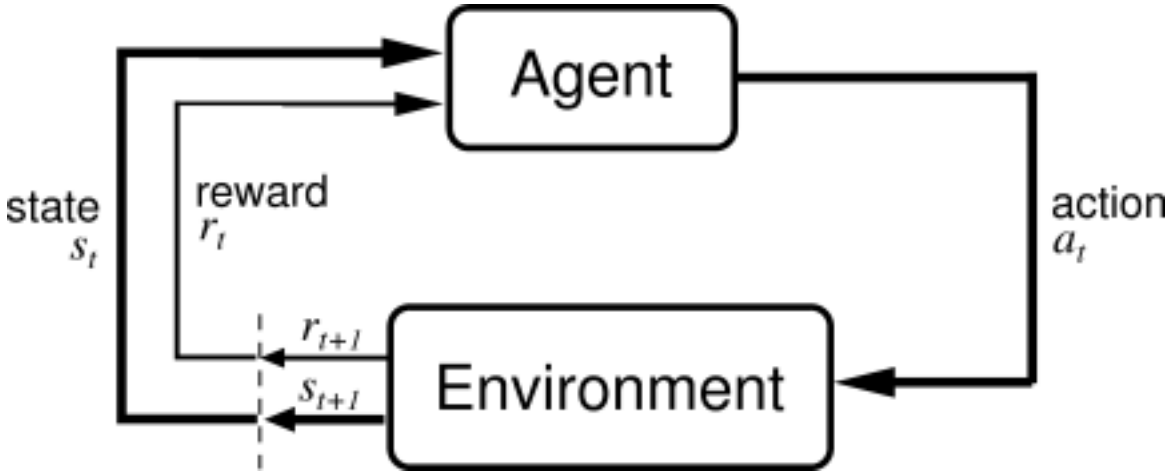
POLICY

REWARD FUNCTION

VALUE FUNCTION

MODEL

AGENT ENVIRONMENT INTERFACE



The **reinforcement learning** problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the **agent**. The thing it interacts with, comprising everything outside the agent, is called the **environment**.

Markov decision processes

Markov decision processes formally describe an environment for reinforcement learning. The environment is fully observable

Markov Property

"The future is independent of the past given the present"

A state St is Markov if and only if

$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots, S_t]$

The state captures all relevant information from the history
Once the state is known, the history may be thrown away.

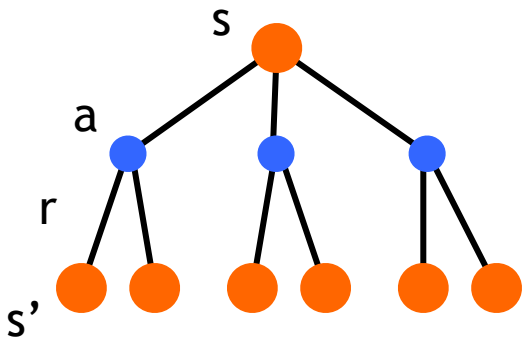
Return

The return Gt
is the total discounted reward from time-step t.
$G_t = R_{t+1} + \gamma R_{t+2} + \ldots =$

The discount $\gamma \in [0, 1]$ is the present value of future rewards
The value of receiving reward R after k + 1 time-steps is $\gamma^{(k)} R$.
This values immediate reward above delayed reward.

Value functions

- state value function: $V^\pi(s)$
  - expected return when starting in *s* and following π
- state-action value function: $Q^\pi(s,a)$
  - expected return when starting in *s*, performing *a*, and following π
- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^\pi(s,a)$
- Bellman equation



$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^\pi(s') \right] = \sum_a \pi(s,a) Q^\pi(s,a)$$

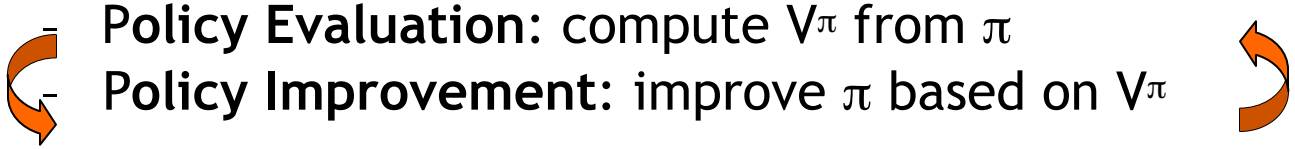
SOLVING AN RL PROBLEM

- **Dynamic Programming**
- **Monte Carlo methods**
- **Temporal-Difference learning**

Dynamic programming

- MAIN IDEA
  - use value functions to structure the search for good policies
  - need a perfect model of the environment

- Two Main components
  - **Policy Evaluation:** compute V^{π} from π
  - **Policy Improvement:** improve π based on V^{π}



  - start with an arbitrary policy
  - repeat evaluation/improvement until convergence

Policy evaluation/improvement

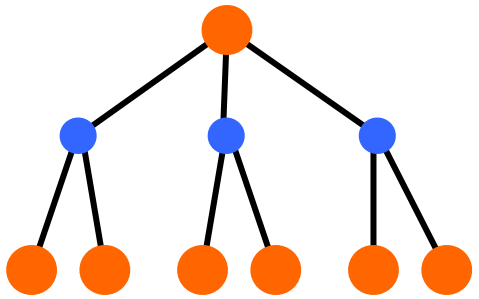
- Policy evaluation: π -> V^{π}
  - start with an arbitrary value function V_0, iterate until V_k converges

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{k'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

- Policy improvement: V^{π} -> π'
  - π' either strictly better than π, or π' is optimal (if $\pi = \pi$')

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

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

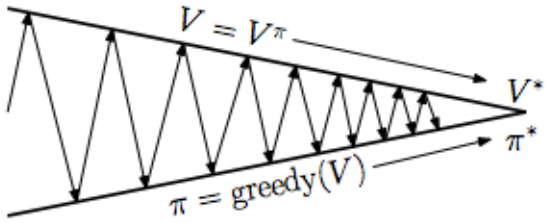


Policy Iteration

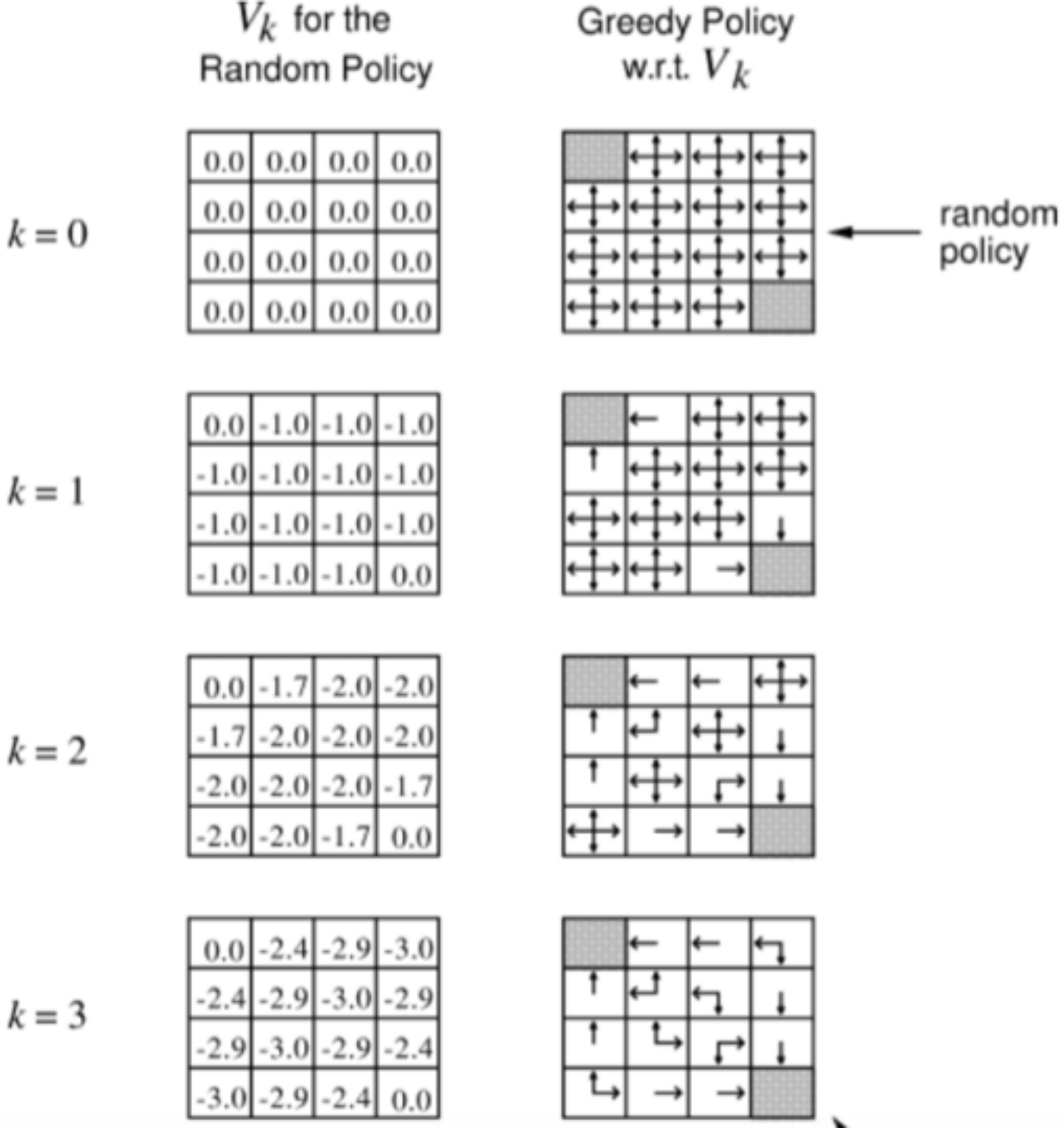
$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \cdots \xrightarrow{E} V^* \xrightarrow{I} \pi^*$$

- E: policy evaluation
- I: policy improvement
- Strict policy improvement at each step
  (unless policy is already optimal)
- Converges often after a few iterations
- Each policy evaluation is itself an iterative process
  ⏩ Can be really slow!

Generalized Policy Iteration (GPI)



- Abstraction/generalization of policy iteration
- Two interacting processes: policy evaluation/improvement
- Update details abstracted away (policy needs to be greedy)
- Don't need to go full way to π or V^π, just "in the direction"
- Both processes converge to a single joint solution (V^*, π^*)


V_k for the Random Policy
Greedy Policy w.r.t. V_k
$k = 0$
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
random policy
$k = 1$
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
$k = 2$
0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0
$k = 3$
0.0 -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 0.0

Monte Carlo methods

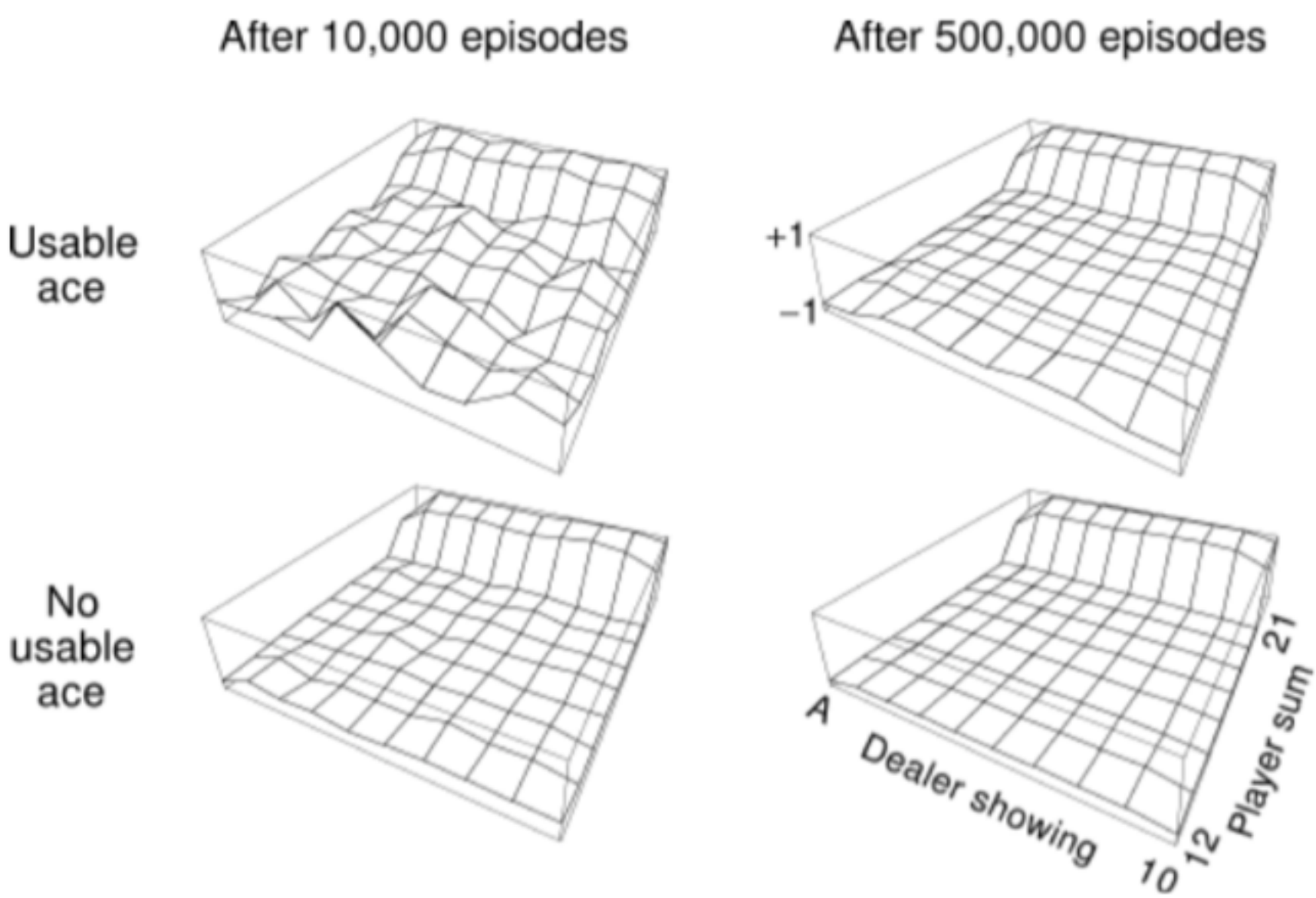
- don’t need full knowledge of environment
  - just experience, or
  - simulated experience
- but similar to DP
  - policy evaluation, policy improvement
- averaging sample returns
  - defined only for episodic tasks

First-Visit Monte Carlo Policy Evaluation

- Generate trajectories with policy π
- Record reward after visiting state s, average at the end

1: **for** $i = 1$ to ∞ **do**
2: Generate trajectory τ_i^π with policy π
3: **for** all states $s \in \tau_i$ **do**
4: $r \leftarrow$ sum of rewards following the first occurrence of s
5: $R(s) \leftarrow [R(s), r]$ ▷ Append r to array
6: **end for**
7: $V^\pi(s) \approx \mathbb{E}[R(s)|\pi] \approx \frac{1}{|R(s)|} \sum_{j=1}^{|R(s)|} R(s)[j]$
8: **end for**

- Convergence to the correct value V^π in the limit


After 10,000 episodes
After 500,000 episodes
Usable ace
No usable ace
+1
−1
A
Dealer showing
10
12
Player sum
21

Temporal-Difference Learning

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

Temporal Difference Learning

- combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: learn from values of successors
  - works for continuous tasks, usually faster than MC

- constant-alpha MC:
  - have to wait until the end of episode to update

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

target

- simplest TD
  - update after every step, based on the successor

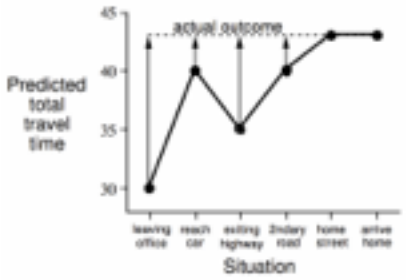
$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

Driving Home Example

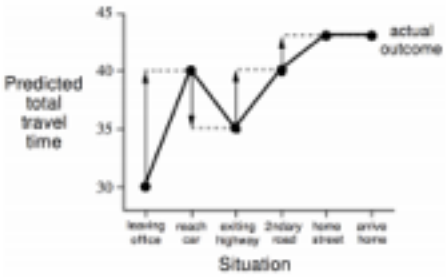
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home Example: MC vs. TD

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)

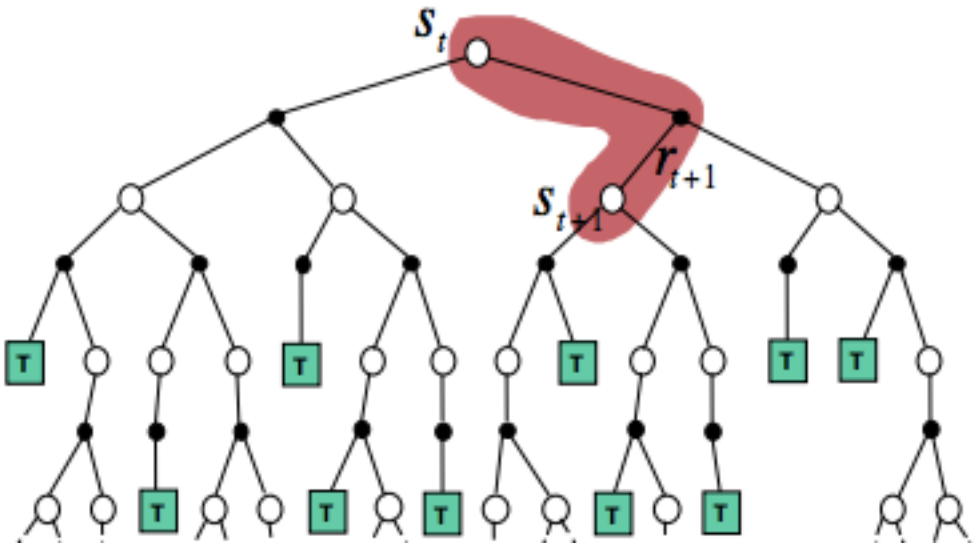


TD(0)

Init.: Set $V(s)$ arbitrarily
repeat for each episode
  $a \leftarrow p_\pi(a|s)$ ▷ Sample action in current state
  Apply a, observe r, s' ▷ Transition to next state
  $V(s) \leftarrow V(s) + \alpha\big(r + \gamma V(s') - V(s)\big)$ ▷ Update V
  $s \leftarrow s'$ ▷ Re-set current state
until s is terminal

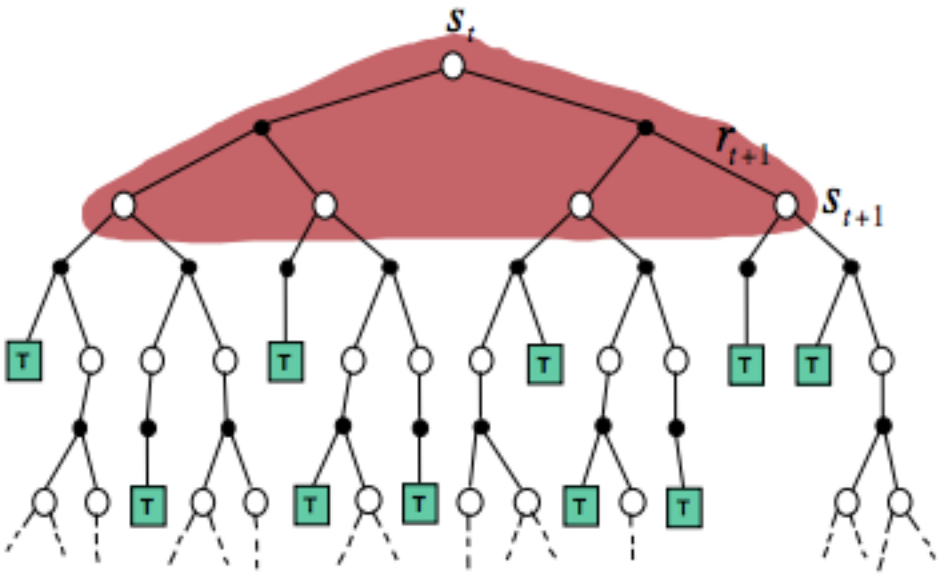
Temporal-Difference Backup

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



Dynamic Programming Backup

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



Monte-Carlo Backup

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