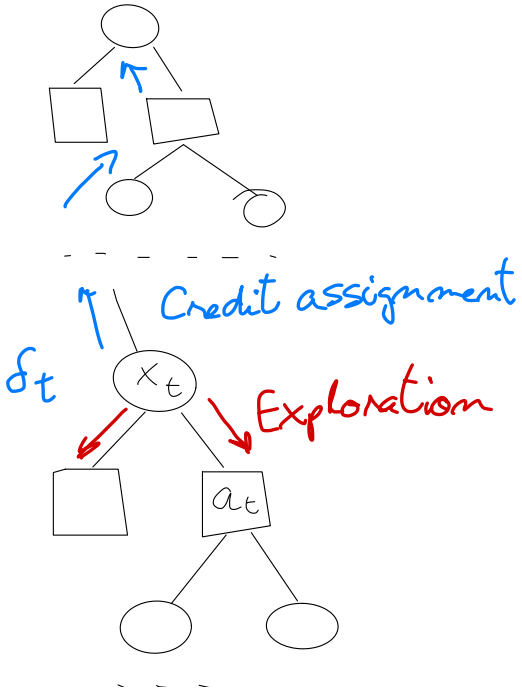
Lecture 6: Markov Decision Processes. Dynamic Programming for policy evaluation

Recall: Sequential decision making

- At time t, agent receives an observation from set $\mathcal{X}$ and can choose an action from set $\mathcal{A}$ (think finite for now)
- Goal of the agent is to maximize long-term return



- Recall the infinite tree of possible interactions of the agent and environment - is finite horizon the only assumption we can make?

Finite clustering assumption

- Paths do NOT need to finish in a finite amount of time
- But, the infinite tree of paths clusters into a finite number of clusters!
- The means *similar situations will recur*
- So we can generalize!

One more step: Markovian assumption

- The way we got to some specific situation is not relevant for the future!
- All that matters is our current observation X_t
- Alternatively, if we should have remembered something, we will consider it part of X_t
- Eg remembering previous image frames or words
- We will call such an observation *state*

Markovian State

- By "the state" at step t, the book means whatever information is available to the agent at step t about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} =$$
$$p(s', r|s, a) = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

- for all $s' \in \mathcal{S}^+, r \in \mathcal{R}$, and all histories $S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t$.

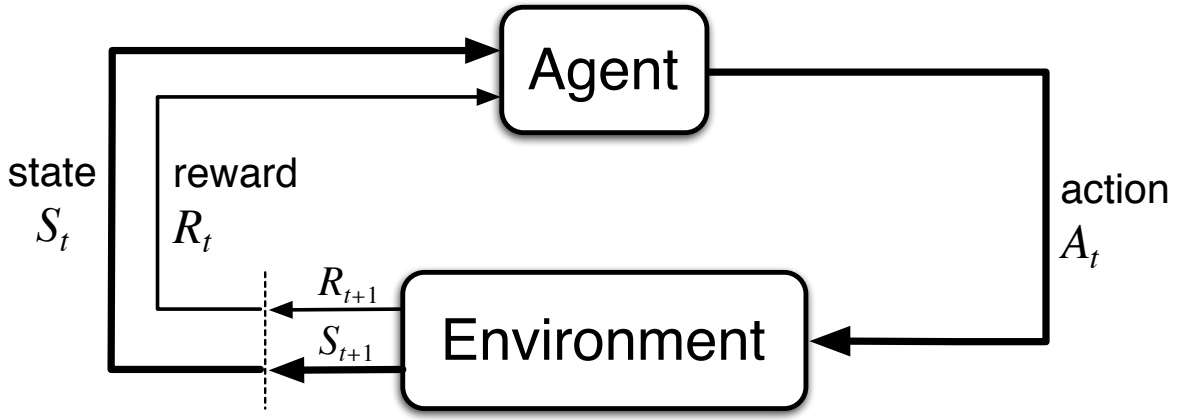
Markov Property

- Next state and reward depend only on the previous state and action, and noting else that happened in the past

$$p(S_{t+1} = s', R_{t+1} = r \,|\, S_t = s, A_t = a) = p(S_{t+1} = s', R_{t+1} = r \,|\, S_t = s, A_t = a, \tau_t), \forall \tau_t$$

- The assumption is useful to develop, analyze and understand algorithms
- It does NOT mean it has to always hold

The Agent-Environment Interface



Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is called a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics"

$$p(s', r|s, a) = \mathbf{Pr}\{S_{t+1}=s', R_{t+1}=r \mid S_t=s, A_t=a\}$$

$$p(s'|s, a) \doteq \Pr\{S_{t+1}=s' \mid S_t=s, A_t=a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a)$$

$$r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$$

An Example Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$

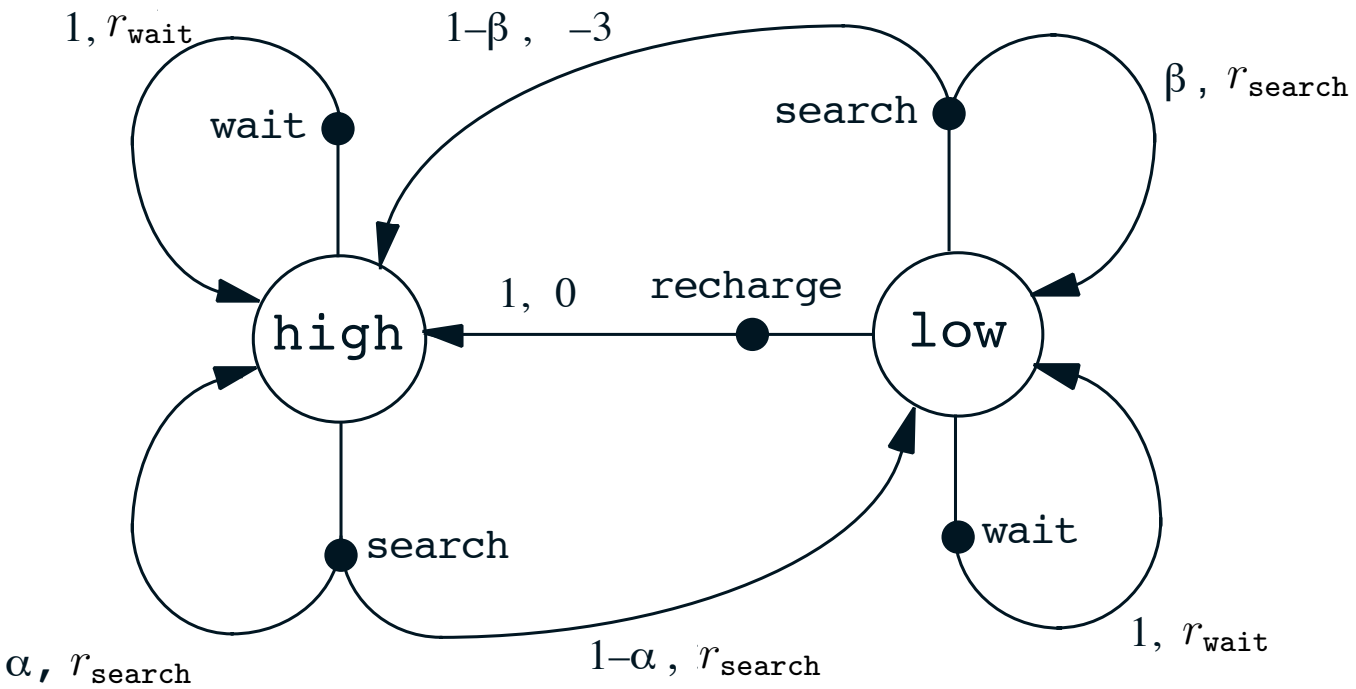
$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$r_{\texttt{search}} > r_{\texttt{wait}}$



Return

Suppose the sequence of rewards after step t is:

$$R_{t+1}, R_{t+2}, R_{t+3}, \ldots$$

What do we want to maximize?

At least three cases, but in all of them,

we seek to maximize the **expected return**, $E\{G_t\}$, on each step t.

- Total reward, G_t = sum of all future reward in the episode
- Discounted reward, G_t = sum of all future *discounted* reward
- Average reward, G_t = average reward per time step

Recall: Episodic Tasks

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze

In episodic tasks, we almost always use simple *total reward*:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T ,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Continuing Tasks

Continuing tasks: interaction does not have natural episodes, but just goes on and on...

In this class, for continuing tasks we will mostly use *discounted return*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

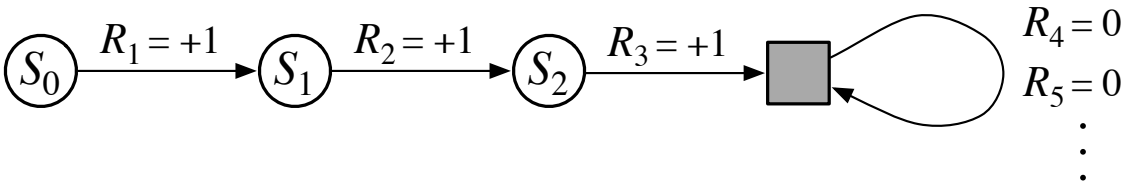
where γ, $0 \le \gamma \le 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

Typically, $\gamma = 0.9$

A Trick to Unify Notation for Returns

- Think of each episode as ending in an absorbing state that always produces reward of zero
- We can cover <u>all</u> cases by writing



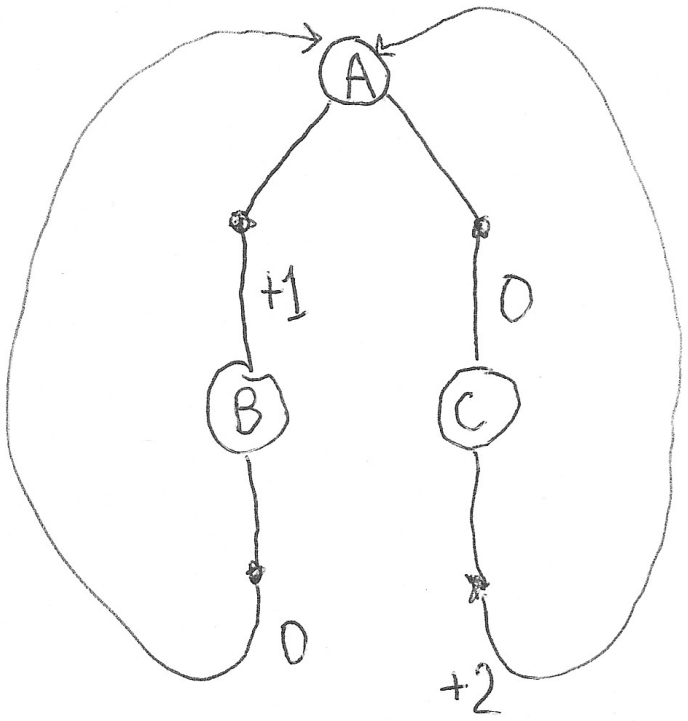
$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ can be 1 only if a zero reward absorbing state is always reached.

Ways to interpret the discount factor

- Inflation!
- Survival probability
- Part of the problem definition, NOT a hyper parameter!!

Example



What policy is optimal?

A: left

B: Right C: other

If $\gamma = 0$?

If $\gamma = .99$

If $\gamma = \frac{1}{2}$?

Episodic and Continuing Tasks: Average Reward

In episodic tasks, we can also use *average reward*:

$$G_0 = (\sum_{t=0}^{T} R_t)/T$$

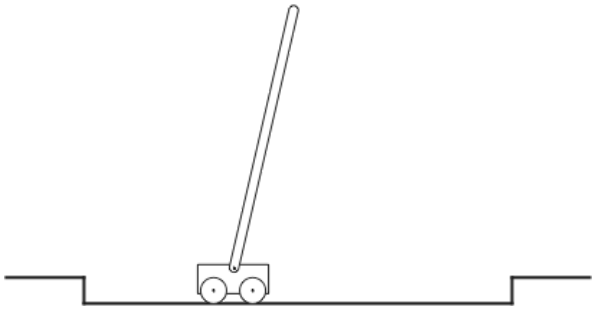
where T is a final time step at which a **terminal state** is reached, ending an episode.

In continuing tasks, we can also define *average reward*:

$$G = \lim_{T \to \infty} \left((\sum_{t=0}^{T} R_t)/T \right)$$

Some advantages and disadvantages compared to discounting

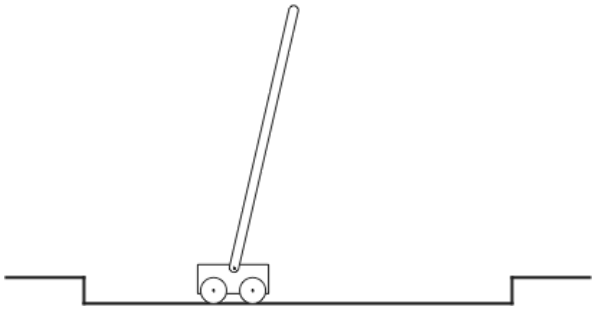
An Example: Pole Balancing



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

As an **episodic task** where episode ends upon failure:

An Example: Pole Balancing



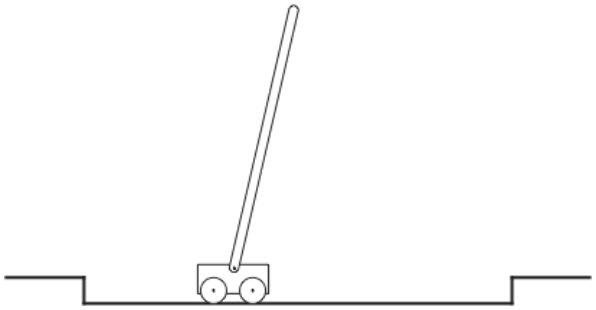
Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

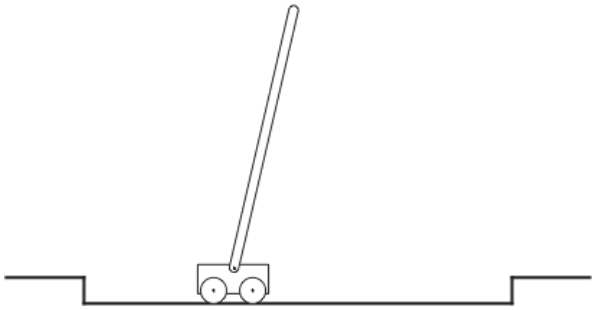
An Example: Pole Balancing



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

As a **continuing task** with discounted return:

An Example: Pole Balancing



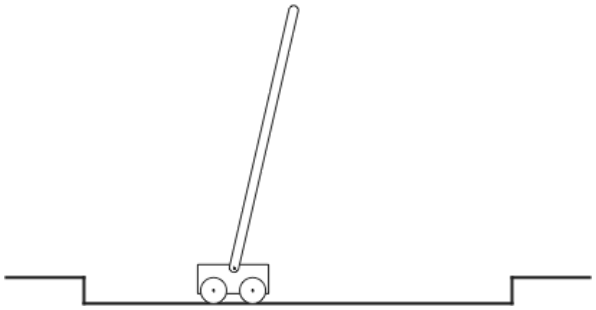
Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

As a **continuing task** with discounted return:

reward = –1 upon failure; 0 otherwise

$\Rightarrow$ return = $-\gamma^k$, for k steps before failure

An Example: Pole Balancing



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

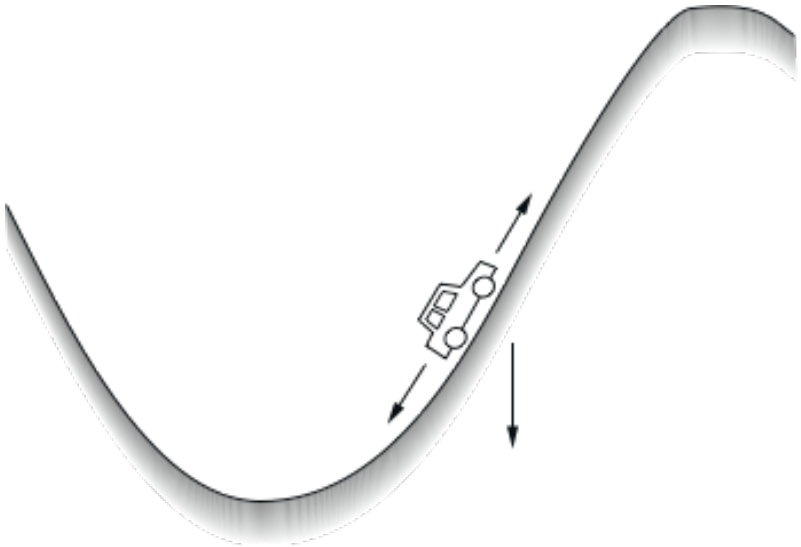
As a **continuing task** with discounted return:

reward = −1 upon failure; 0 otherwise

$\Rightarrow$ return = $-\gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

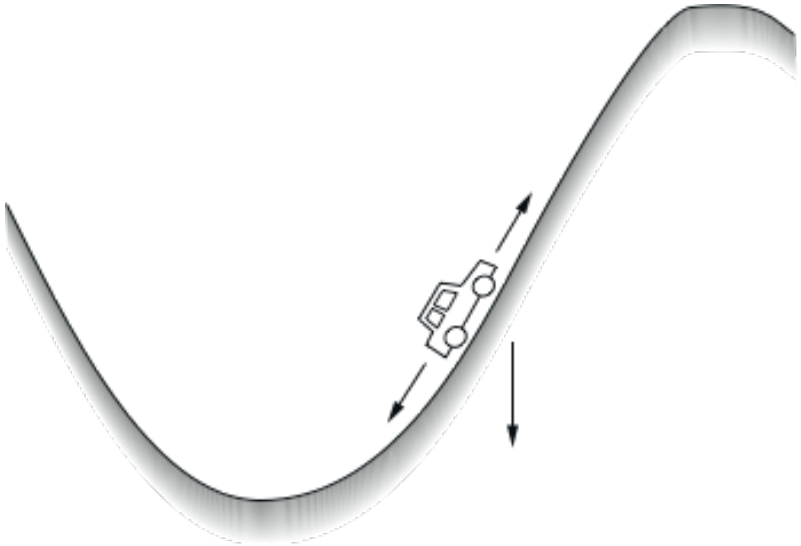
Another Example: Mountain Car



Get to the top of the hill
as quickly as possible.

Return is maximized by minimizing
number of steps to reach the top of the hill.

Mountain Car: Discounted

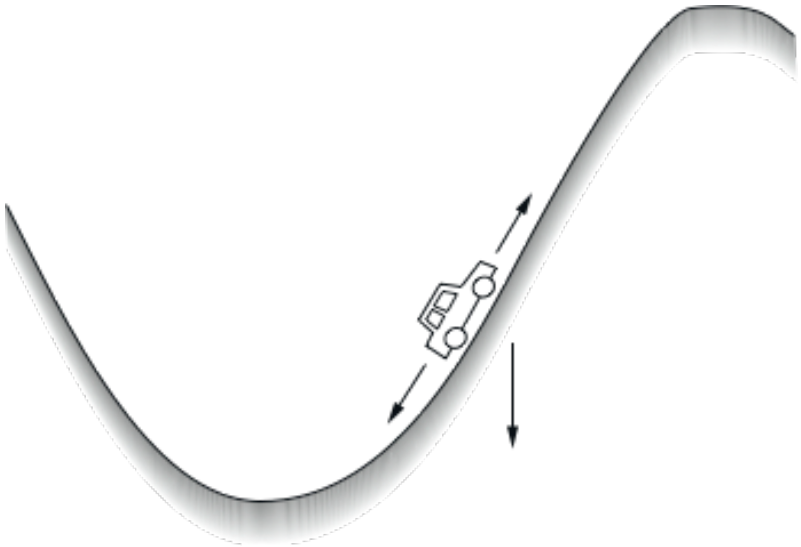


Get to the top of the hill
as quickly as possible.

Reward: 1 at the top of the hill, 0 otherwise
Return: if discount <1, k=number of time steps, so return is γ^k

Return is maximized by minimizing
number of steps to reach the top of the hill.

Mountain Car: Episodic



Get to the top of the hill
as quickly as possible.

reward = –1 for each step where **not** at top of hill

$\Rightarrow$ return = – number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State - value function for policy π :

$$v_\pi(s) = E_\pi\left\{G_t \mid S_t = s\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right\}$$

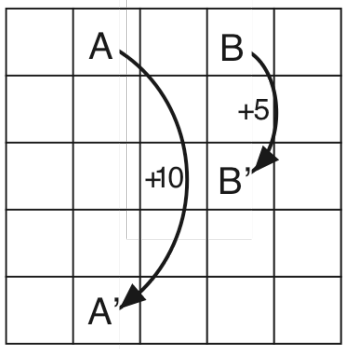
- The **value of an action (in a state)** is the expected return starting after taking that action from that state; depends on the agent's policy:

Action - value function for policy π :

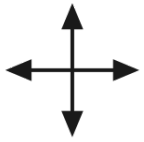
$$q_\pi(s,a) = E_\pi\left\{G_t \mid S_t = s, A_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right\}$$

Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



(a)



Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

State-value function for equiprobable random policy; $\gamma = 0.9$

Bellman Equation for a Policy π

The basic idea:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

So:

$$\begin{aligned} v_\pi(s) &= E_\pi \left\{ G_t \,\middle|\, S_t = s \right\} \\ &= E_\pi \left\{ R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \middle|\, S_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

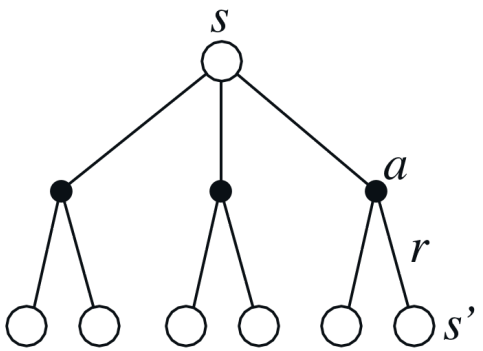
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right]$$

More on the Bellman Equation

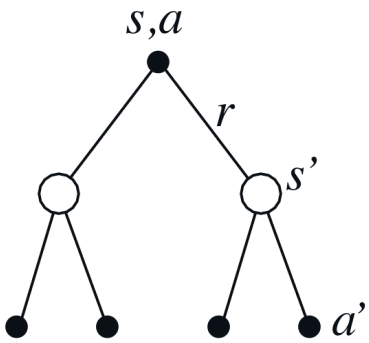
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]$$

This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

Backup diagrams:



for v_π



for q_π

Iterative Methods

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

a "sweep"

A sweep consists of applying a **backup operation** to each state.

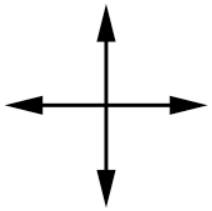
A **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big] \qquad \forall s \in \mathcal{S}$$

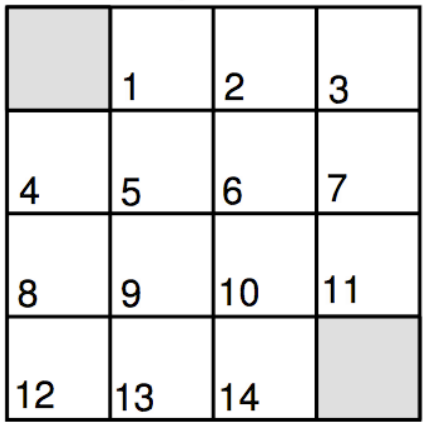
Iterative Policy Evaluation – One array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

A Small Gridworld



actions



$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

Iterative Policy Eval for the Small Gridworld

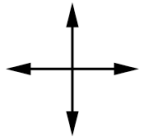
V_k for the Random Policy

$k = 0$

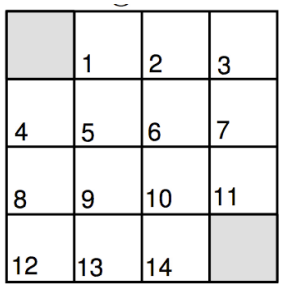
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$\pi =$ equiprobable random action choices

$k = 1$



actions



$R = -1$
on all transitions

$\gamma = 1$

$k = 2$

$k = 3$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

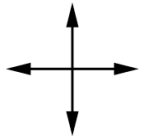
$k = 10$

$k = \infty$

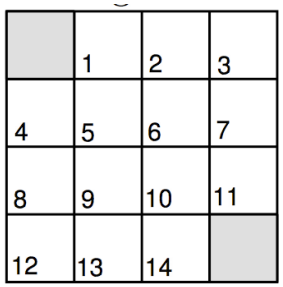
Iterative Policy Eval for the Small Gridworld

V_k for the Random Policy

π = equiprobable random action choices



actions



$R = -1$
on all transitions

$\gamma = 1$

$k = 0$

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

$k = 1$

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

$k = 2$

$k = 3$

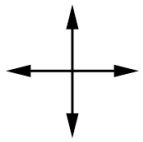
- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

$k = 10$

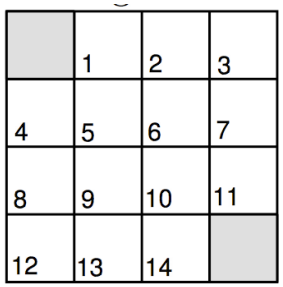
$k = \infty$

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



actions



$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

V_k for the Random Policy

$k = 0$

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

$k = 1$

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

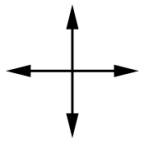
$k = 3$

$k = 10$

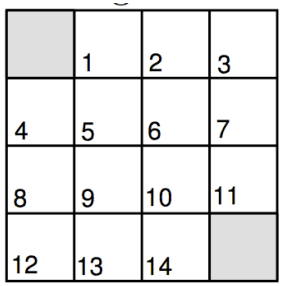
$k = \infty$

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



actions



$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

V_k for the Random Policy

$k = 0$

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

$k = 1$

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

$k = 3$

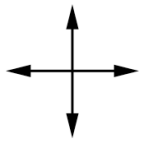
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

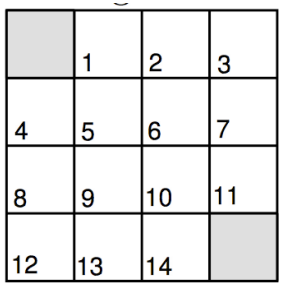
$k = \infty$

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



actions



$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

V_k for the Random Policy

$k = 0$

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

$k = 1$

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

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Asynchronous DP

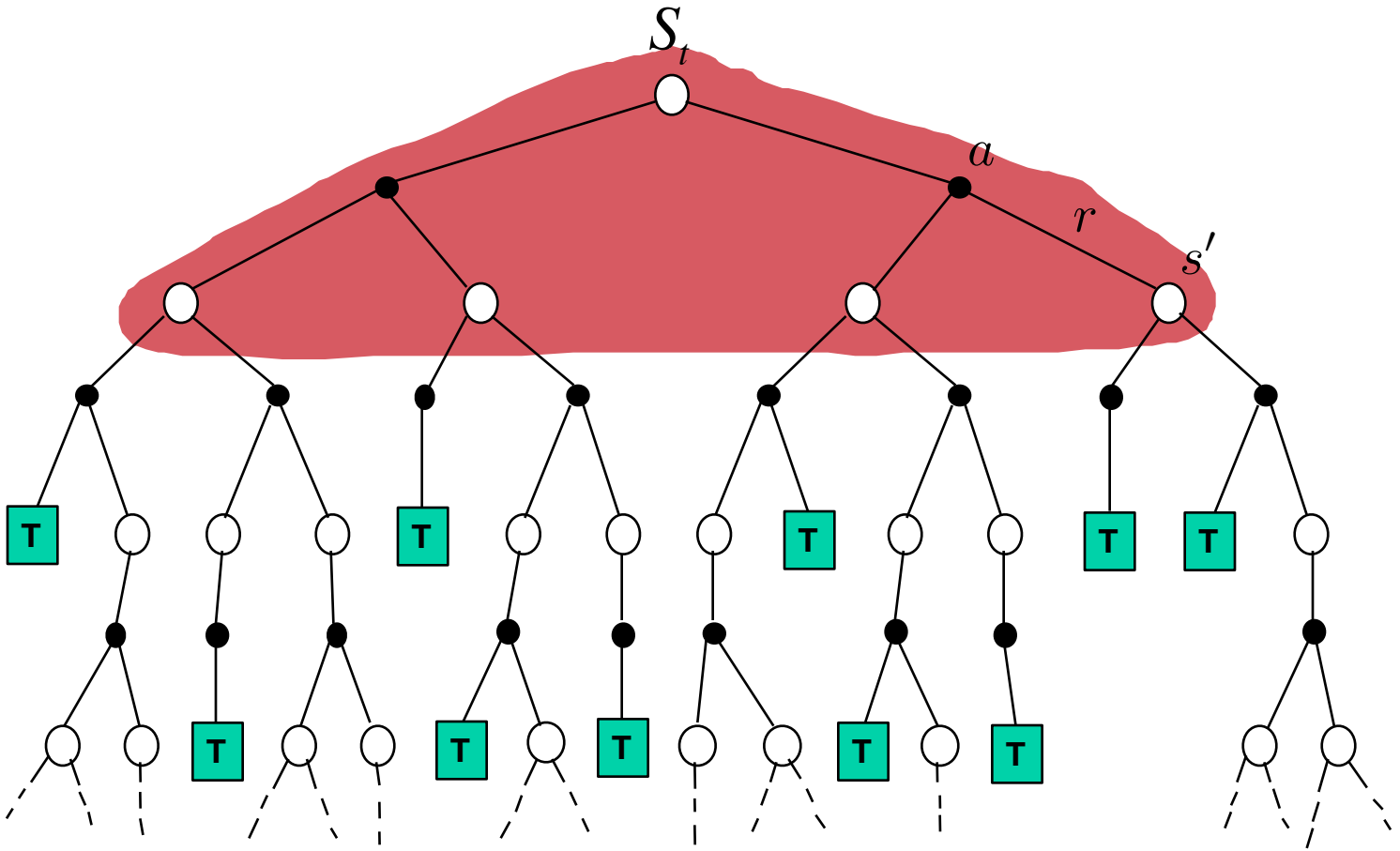
- Finding the value function of a policy implies solving a linear system of equations
- So complexity is polynomial (cubic, or possibly quadratic) in the number of states
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- In practice, classical DP can be applied to problems with a few millions of states, not more
- Asynchronous DP can be applied to larger problems, and is appropriate for parallel computation.

Asynchronous DP

- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Pick a state at random and apply the appropriate backup
- Still need lots of computation, but does not get locked into hopelessly long sweeps
- Can you select states to backup intelligently? YES: an agent's experience can act as a guide.

Asynchronous DP Policy Evaluation

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')]$$

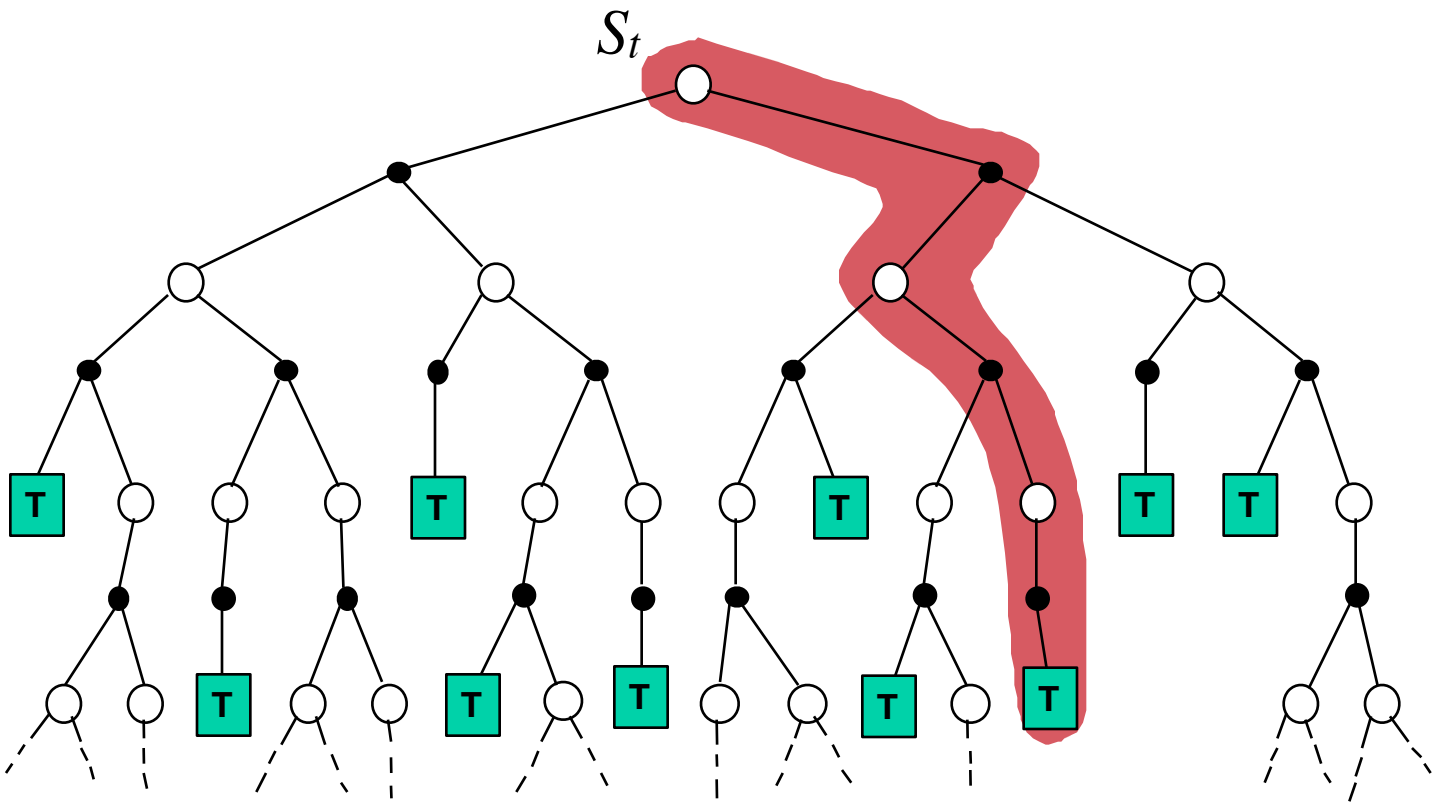


DP methods

- Require a model of the transition dynamics and rewards
- If we don't know it, we need to learn it from data!
- *Model-based RL methods* learn a model from the data then use it to do approximate asynchronous DP
- But learning a model can be expensive!
- MC did not need a model! Just trajectories

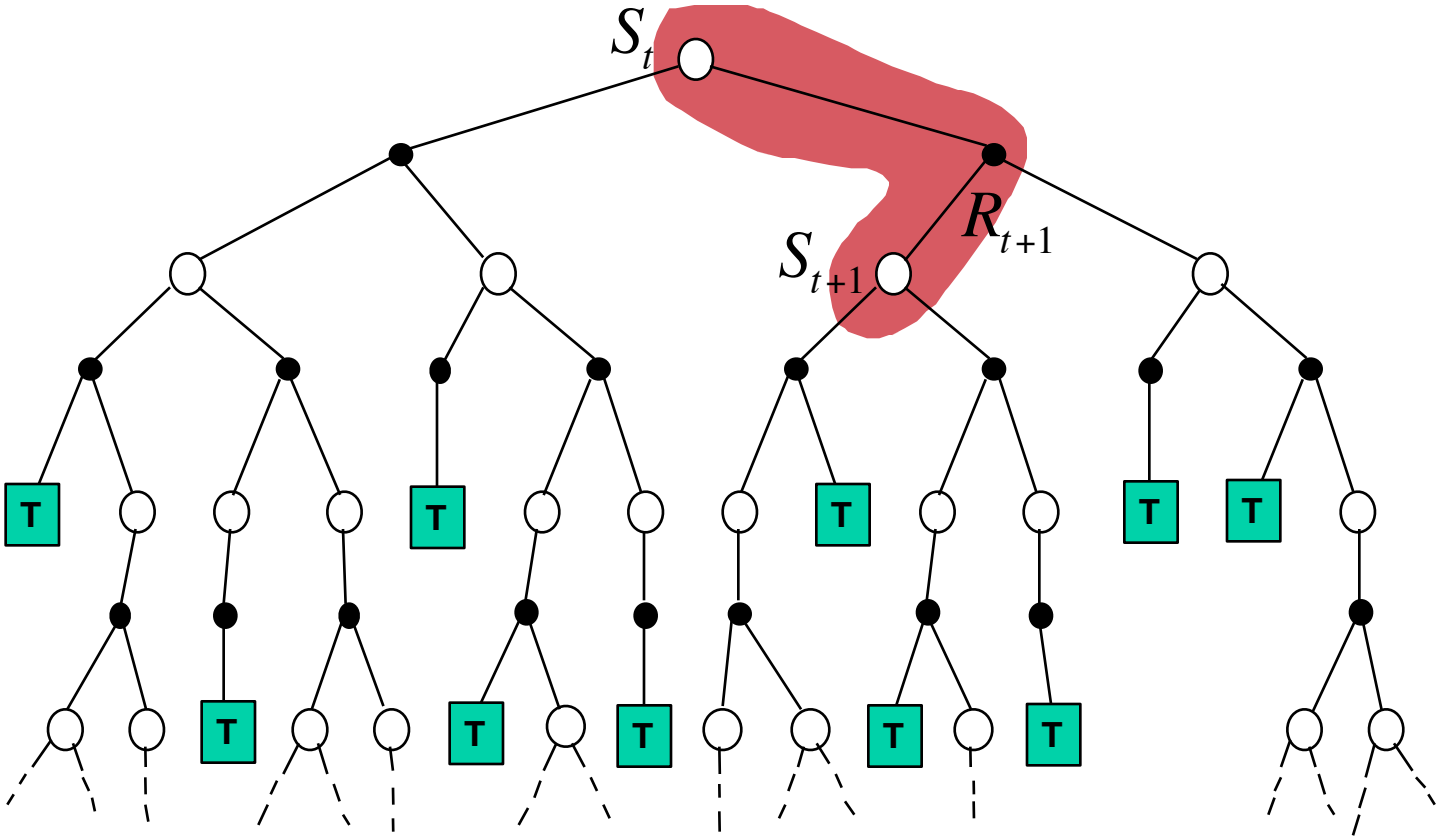
Recall: Simple Monte Carlo

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



Temporal-Difference Learning: Between MC and DP!

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



Temporal-Difference (TD) Prediction

Policy Evaluation (the prediction problem):
for a given policy π, compute the state-value function v_π

Simple every-visit Monte Carlo method:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

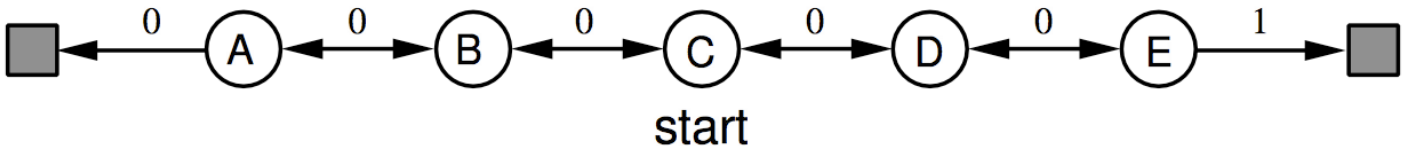
target: the actual return after time *t*

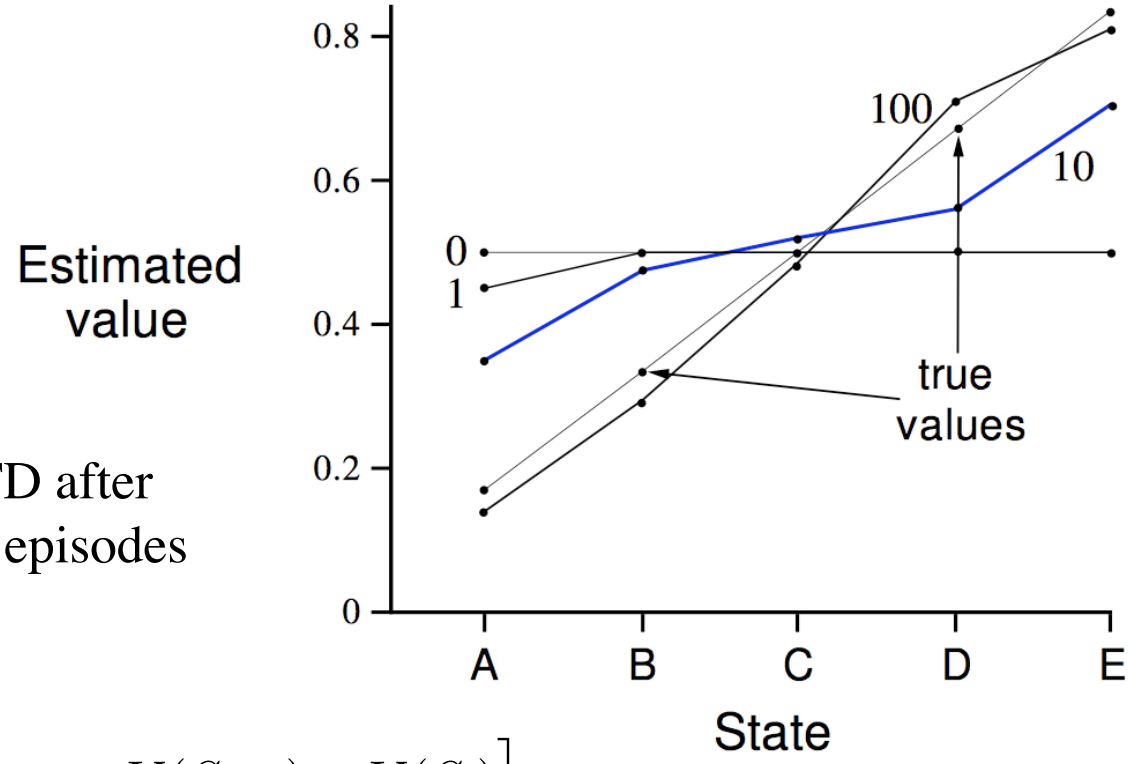
The simplest temporal-difference method TD(0):

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

target: an estimate of the return

Random Walk Example

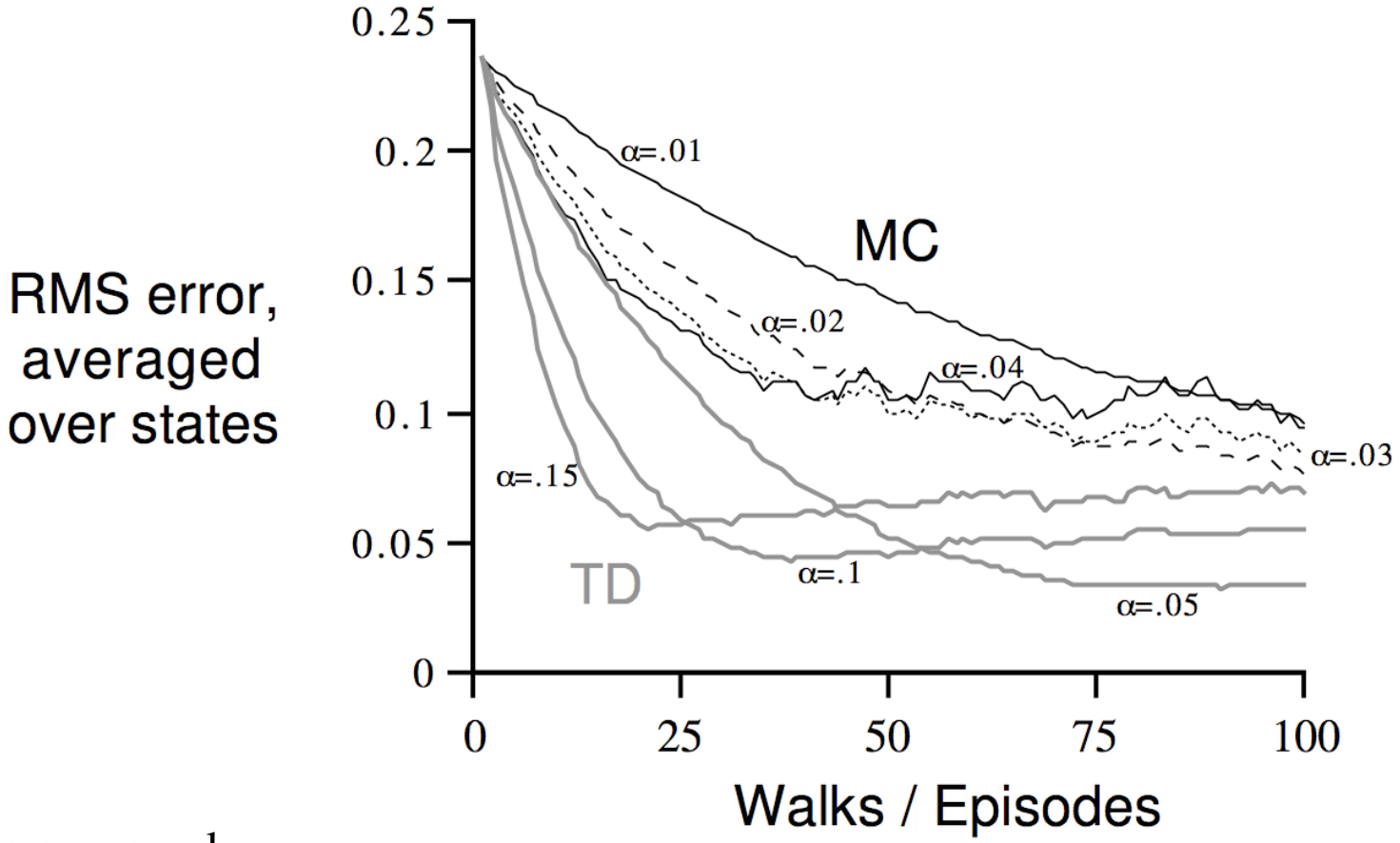




Values learned by TD after various numbers of episodes

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

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes