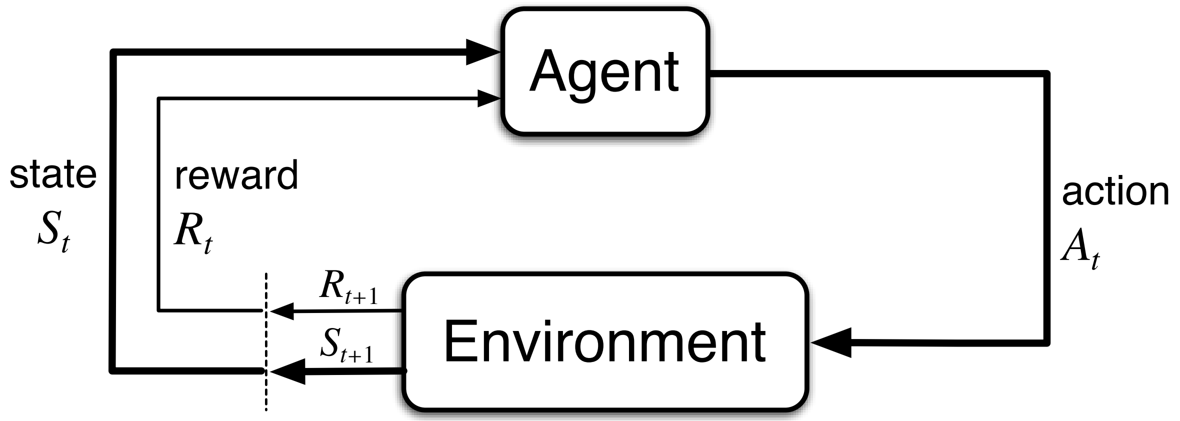
Sequential decision making
Monte Carlo Policy Evaluation
Temporal-Difference Learning

Recall: 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: $S_{t+1} \in \mathcal{S}^+$

Recall: Policy Evaluation

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

Recall: **State-value function for policy π**

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Recall: **Bellman equation for v_π**

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

—a system of $|\mathcal{S}|$ simultaneous equations

Recall: Iterative Methods

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

a “sweep”

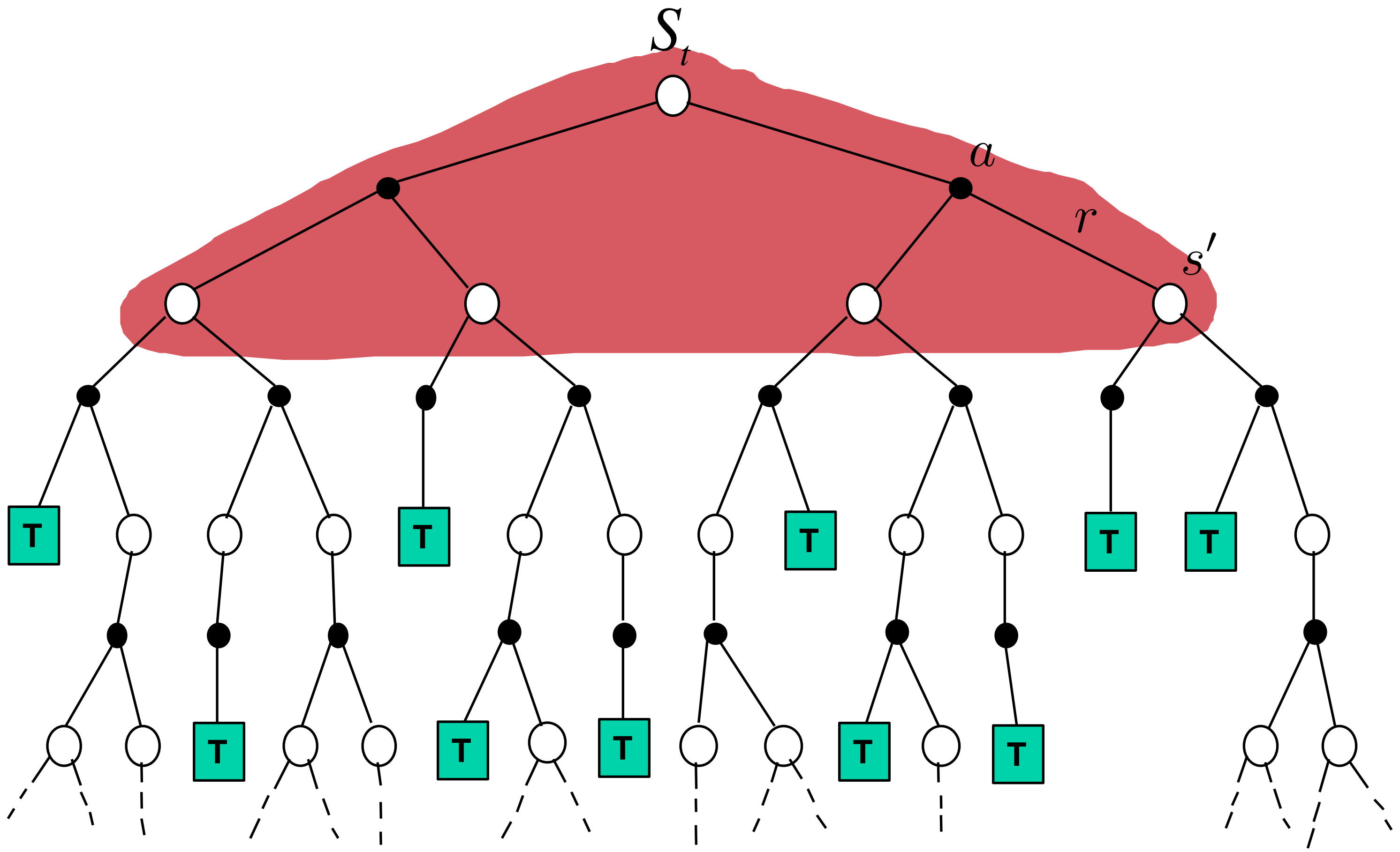
A sweep consists of applying a **backup operation** to each state.

A **full policy-evaluation backup**:

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

Dynamic Programming Policy Evaluation

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

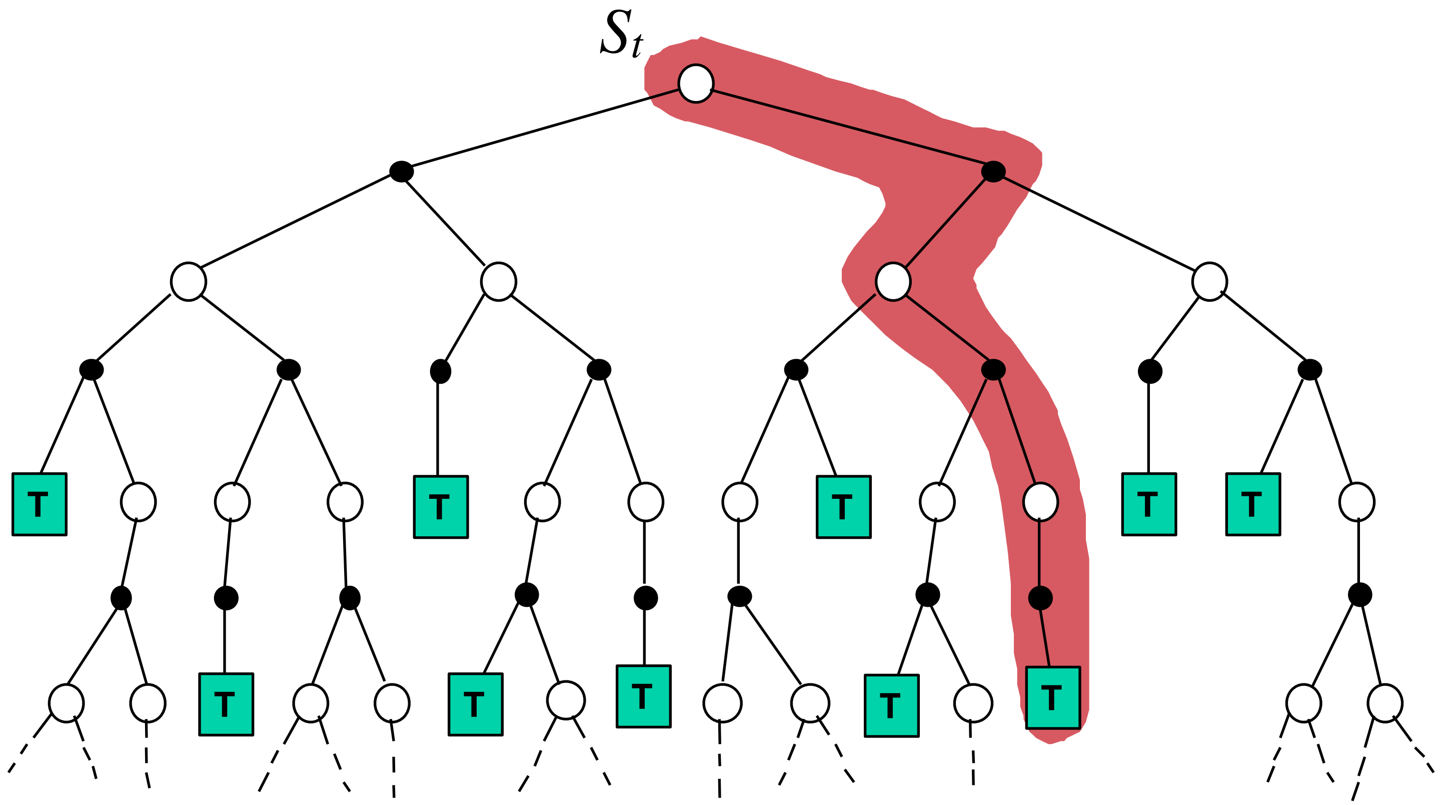


From Planning to Learning

- DP requires a *probability model* (as opposed to a generative or simulation model)
- We can interact with the world, learning a model (rewards and transitions) and then do DP
- This approach is called model-based RL
- Full probability model may hard to learn though
- Today: direct learning of the value function from interaction
- Still focusing on evaluating a fixed policy

Simple Monte Carlo

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

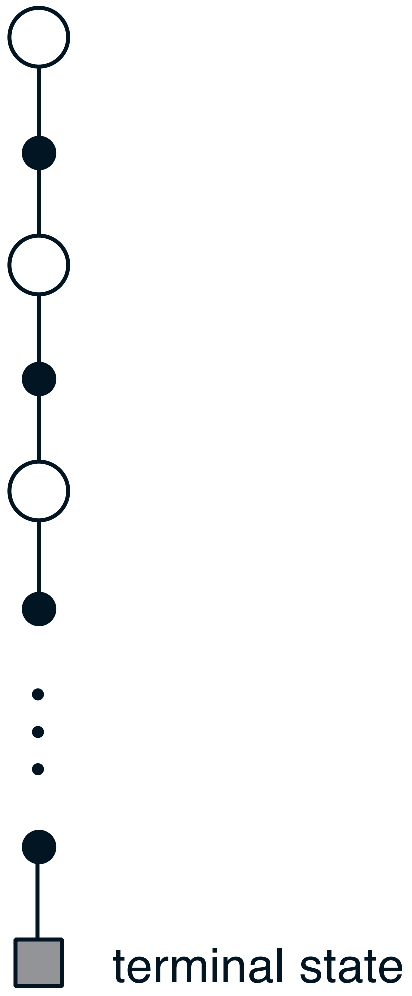


Monte Carlo Methods

- Monte Carlo methods are learning methods
  Experience → values, policy
- Monte Carlo methods can be used in two ways:
  - *model-free:* No model necessary and still attains optimality
  - *simulated:* Needs only a simulation, not a *full* model
- Monte Carlo methods learn from *complete* sample returns
  - Defined for episodic tasks (in the book)
- Like an associative version of a bandit method

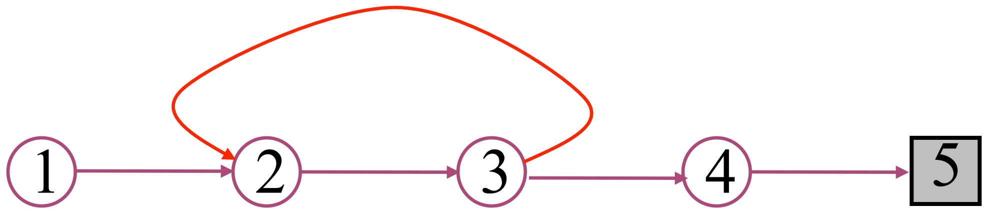
Backup diagram for Monte Carlo

- Entire rest of episode included
- Only one choice considered at each state (unlike DP)
  - thus, there will be an explore/exploit dilemma
- Does not bootstrap from successor states's values (unlike DP)
- Time required to estimate one state does not depend on the total number of states



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

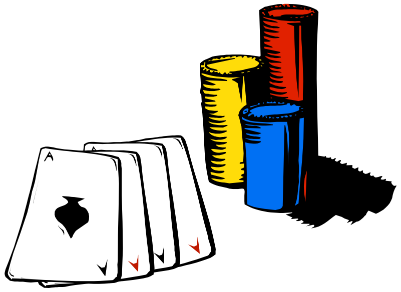
MC vs supervised regression

- Target returns can be viewed as a supervised label (true value we want to fit)
- State is the input
- We can use any function approximator to fit a function from states to returns! Neural nets, linear, nonparametric…

- *Unlike supervised learning: there is strong correlation between inputs and between outputs!*
- Due to the lack of iid assumptions, theoretical results from supervised learning cannot be directly applied

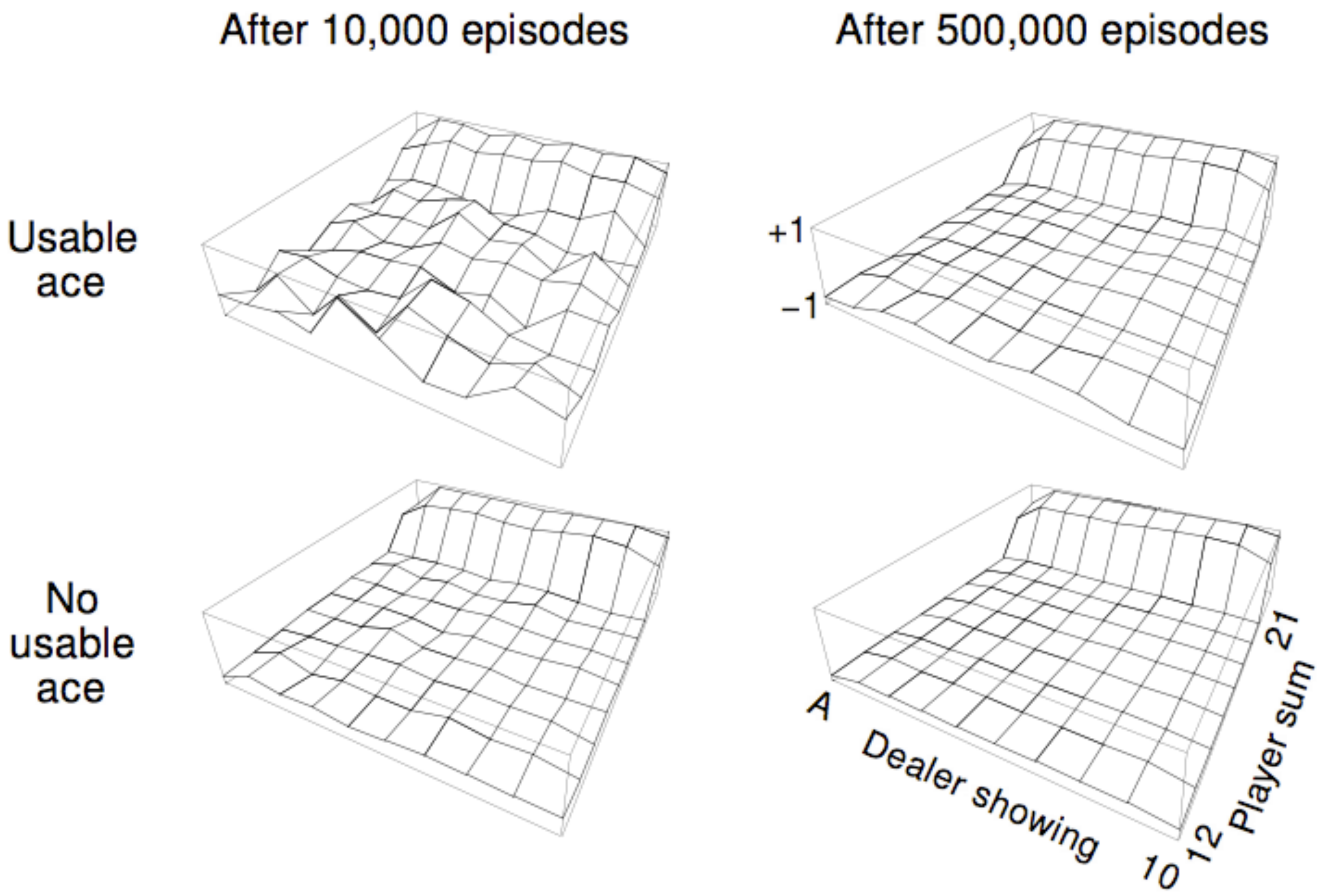
Blackjack example

- *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?



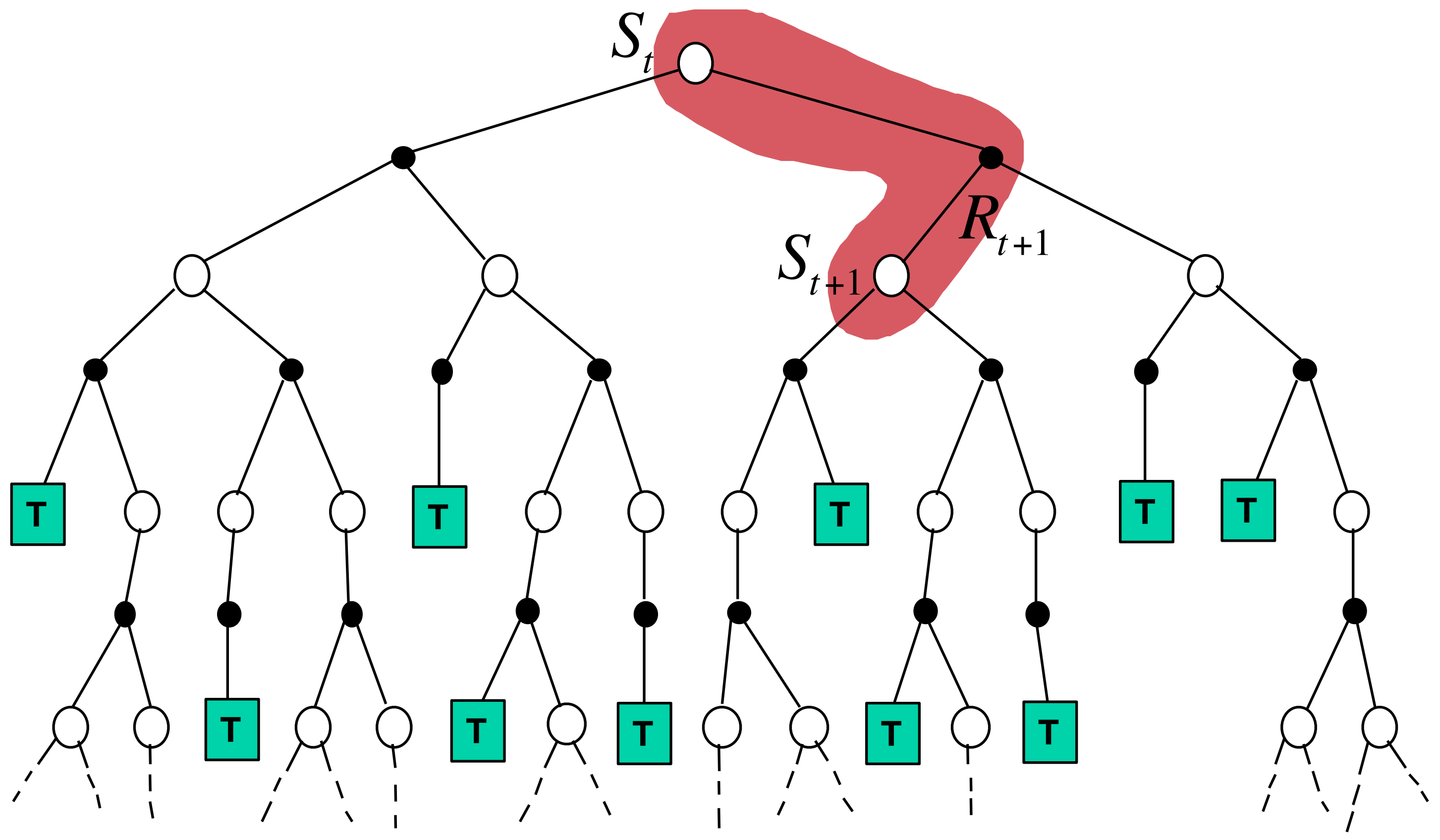
- *Reward:* +1 for winning, 0 for a draw, -1 for losing
- *Actions:* stick (stop receiving cards), hit (receive another card)
- *Policy:* Stick if my sum is 20 or 21, else hit
- No discounting ($\gamma = 1$)

Learned blackjack state-value functions



Simplest TD Method

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



TD methods bootstrap and sample

- Bootstrapping: update involves an *estimate*
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an *expected value*
  - MC samples
  - DP does not sample
  - TD samples

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:

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[\underset{\uparrow}{G_t} - V(S_t) \Big]$$

target: the actual return after time *t*

The simplest temporal-difference method TD(0):

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

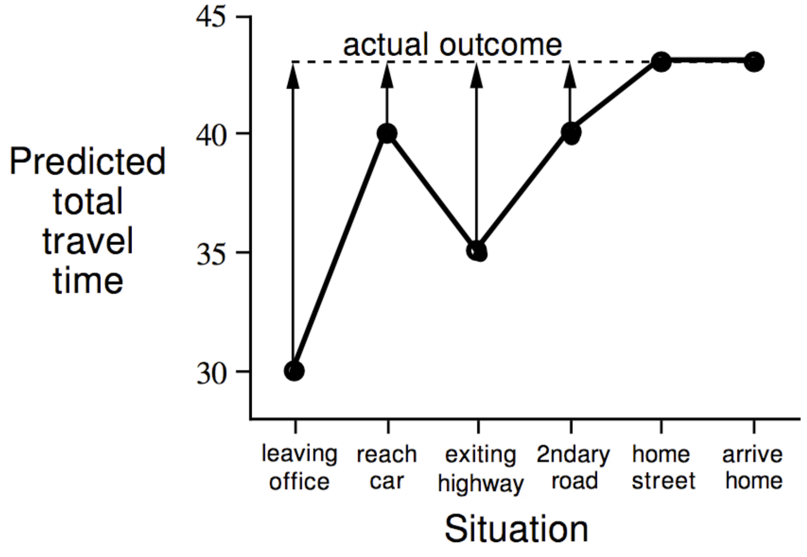
target: an estimate of the return

Example: Driving Home

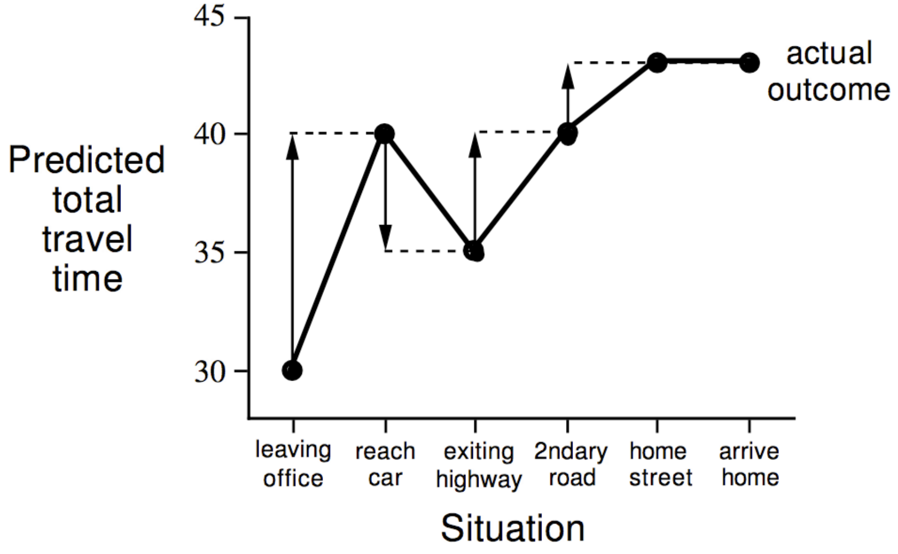
State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)



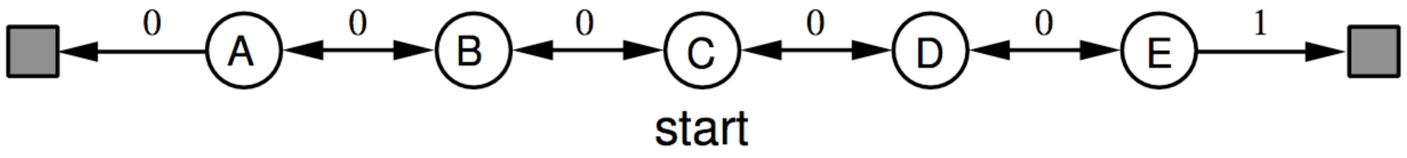
Changes recommended by TD methods (α=1)

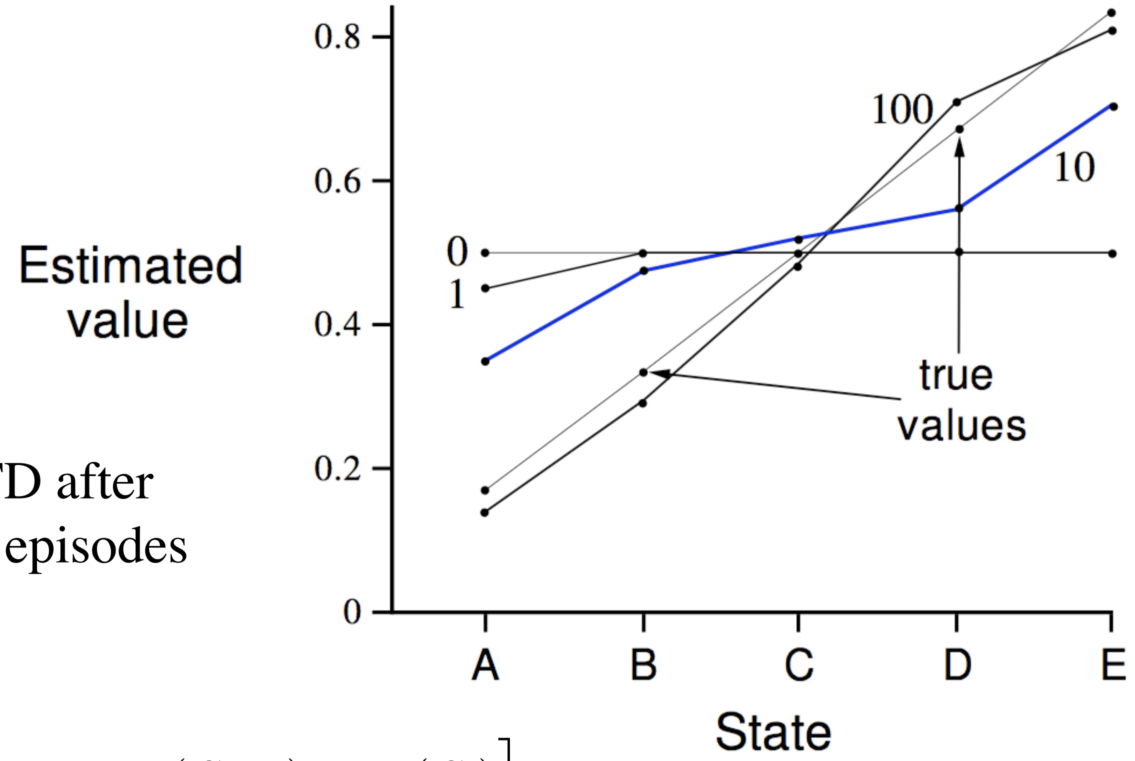


Advantages of TD Learning

- 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
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Random Walk Example

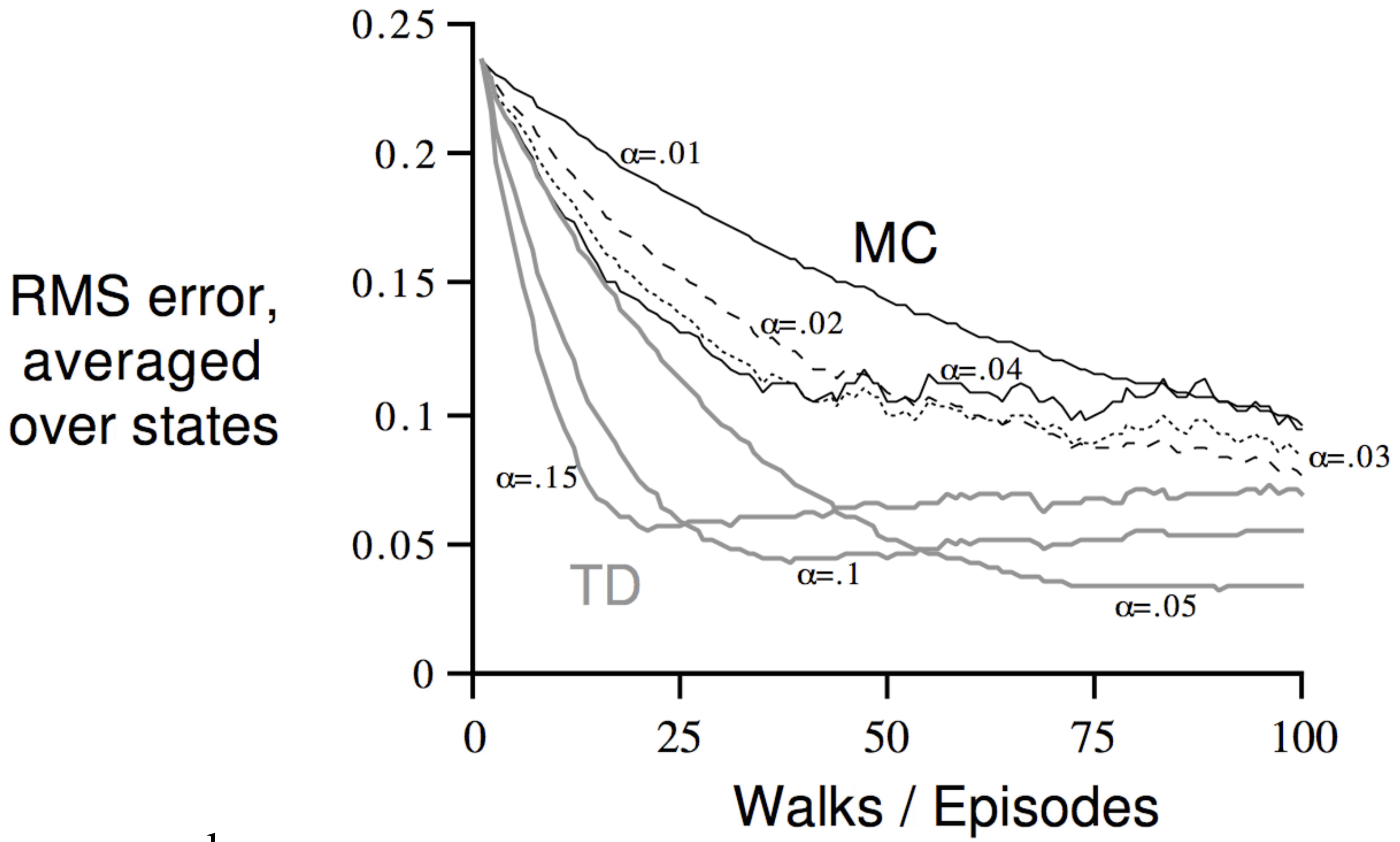




Values learned by TD after various numbers of episodes

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

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Batch Updating in TD and MC methods

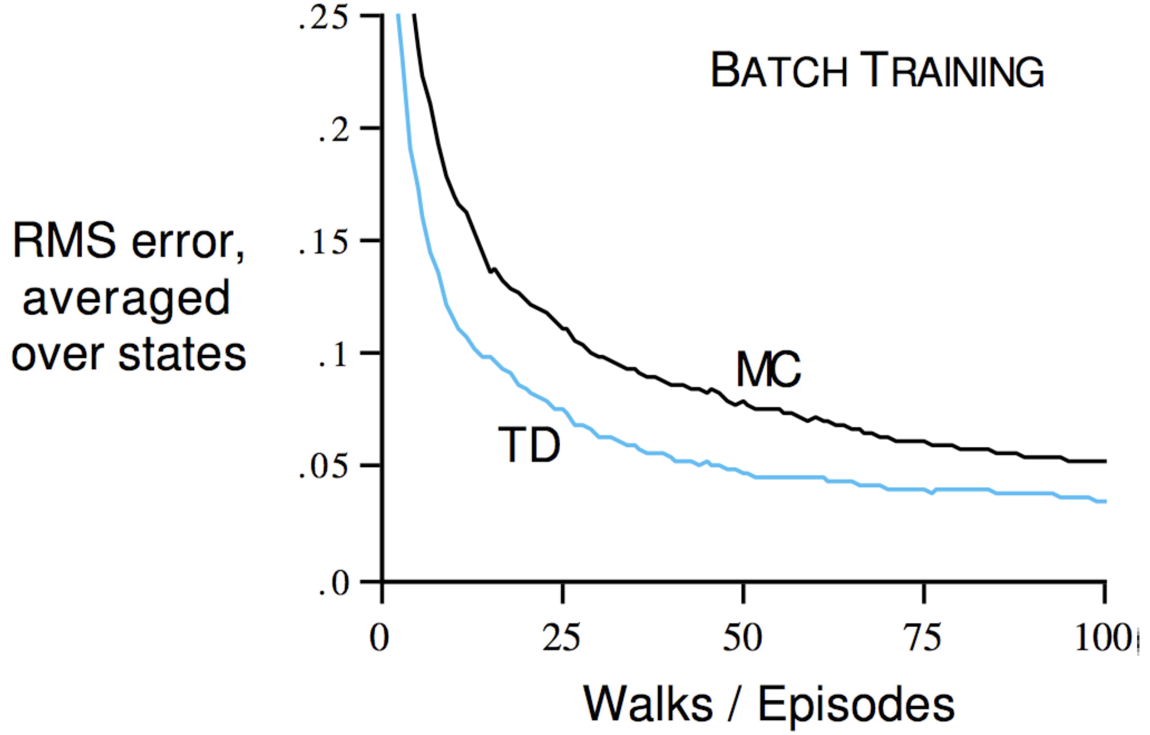
Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.

Compute updates according to TD or MC, but only update estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating, TD converges for sufficiently small α.

Constant-α MC also converges under these conditions, but to a different answer!

Random Walk under Batch Updating



After each new episode, all previous episodes were treated as a batch, and algorithm was trained until convergence. All repeated 100 times.

You are the Predictor

Suppose you observe the following 8 episodes:

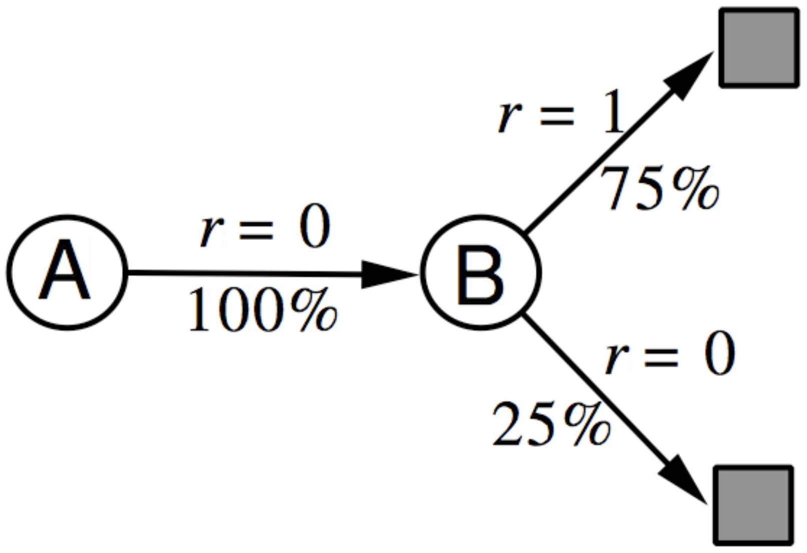
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

$V(\mathrm{B})$?

$V(\mathrm{A})$?

Assume Markov states, no discounting ($\gamma = 1$)

You are the Predictor

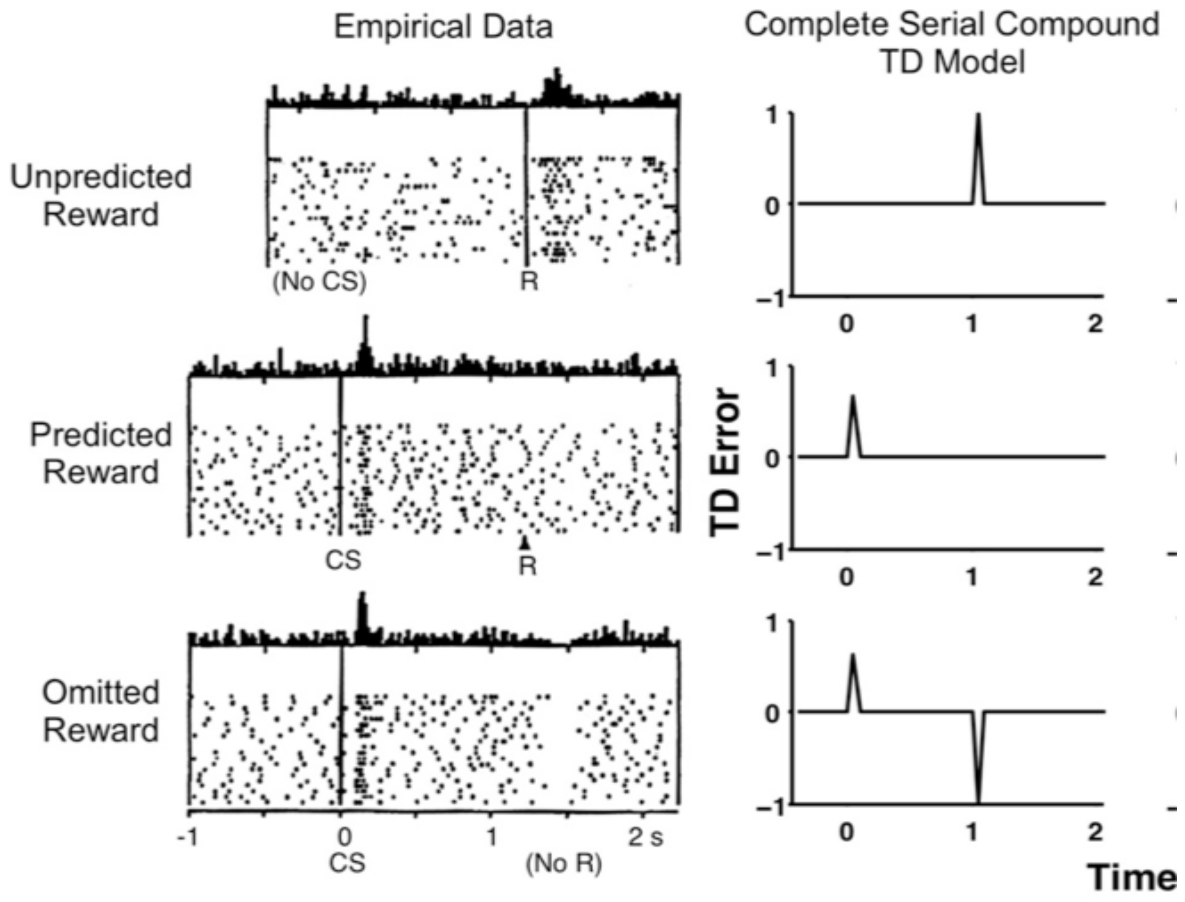


$V(\mathrm{A})?$

You are the Predictor

- The prediction that best matches the training data is V(A)=0
  - This minimizes the mean-square-error on the training set
  - This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set V(A)=.75
  - This is correct for the maximum likelihood estimate of a Markov model generating the data
  - i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts (how?)
  - This is called the certainty-equivalence estimate
  - This is what TD gets

Application of TD
Dopamine neuron activity modelling

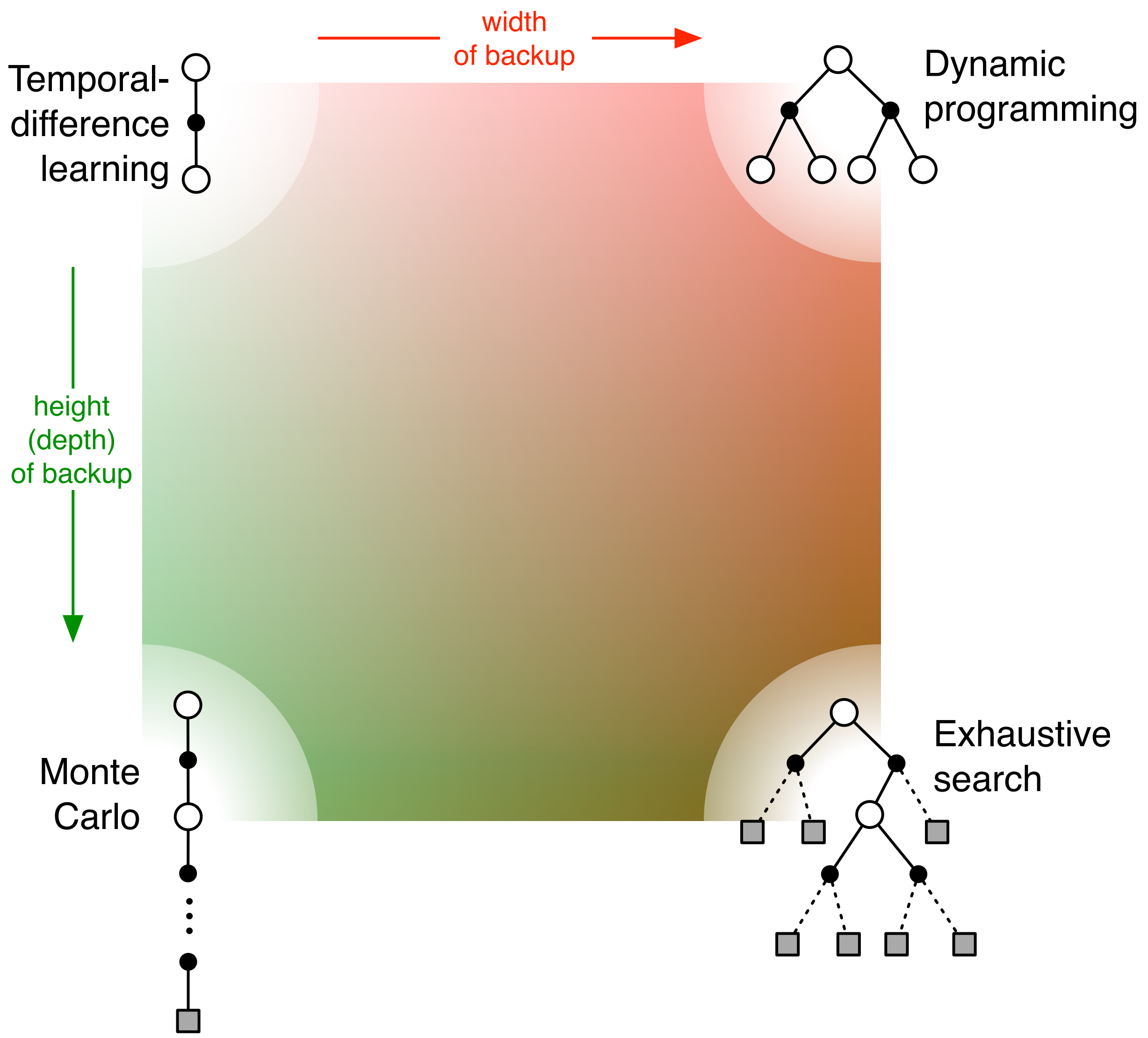


Cf. Shultz, Dayan et al, 1996; and lots of follow-up work including MNI, Psych.

Summary so far

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data

Unified View


width
of backup
Temporal-
difference
learning
Dynamic
programming
height
(depth)
of backup
Monte
Carlo
Exhaustive
search