Lecture 7: Temporal-Difference Learning. More on dynamic programming

Administrative items

- Assignment-1 is **due tomorrow**
- Assignment-2 will be released tomorrow and the deadline is in two weeks from the release time (Feb 4th, 2026)

Recall: 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*

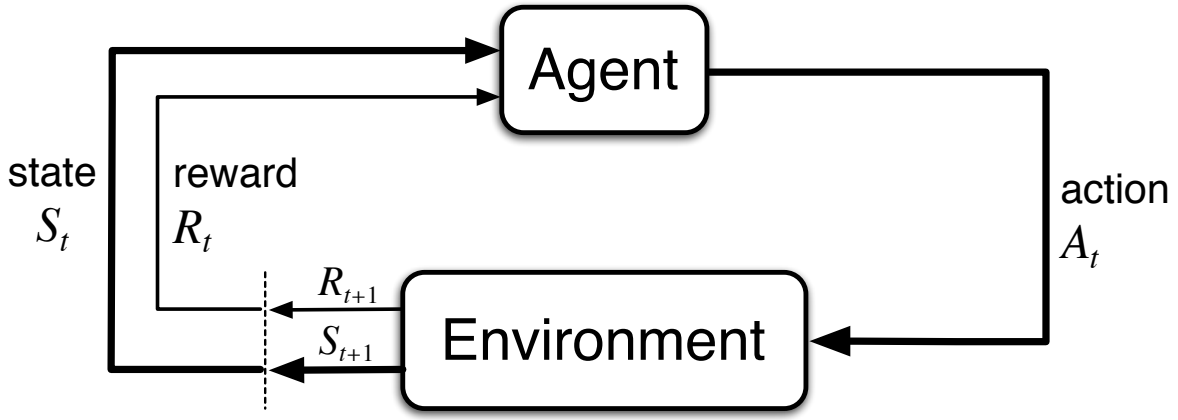
Recall: Markov Property

- Next state and reward depend only on the previous state and action, and nothing 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: $S_t \in \mathcal{S}$

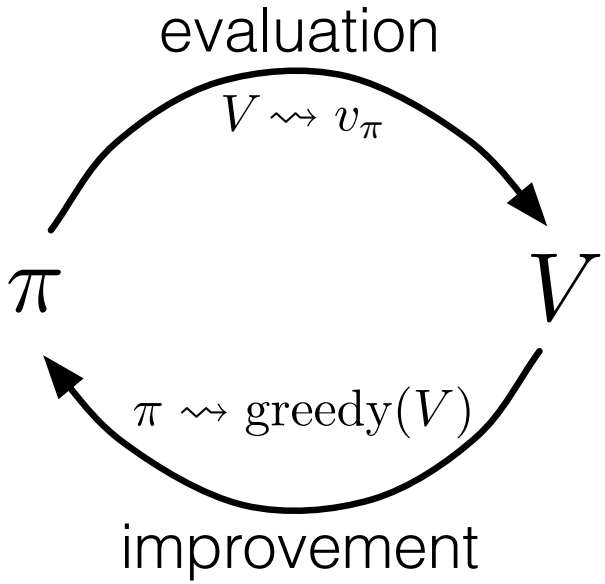
produces action at step t : $A_t \in \mathcal{A}(S_t)$

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

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

- **Goal of the agent is to maximize its expected returns**

Generalized Policy Iteration



- **Policy evaluation or Prediction:** Estimate the value function of a policy π.

Recall: 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"

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

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

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

Recall: 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 :

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

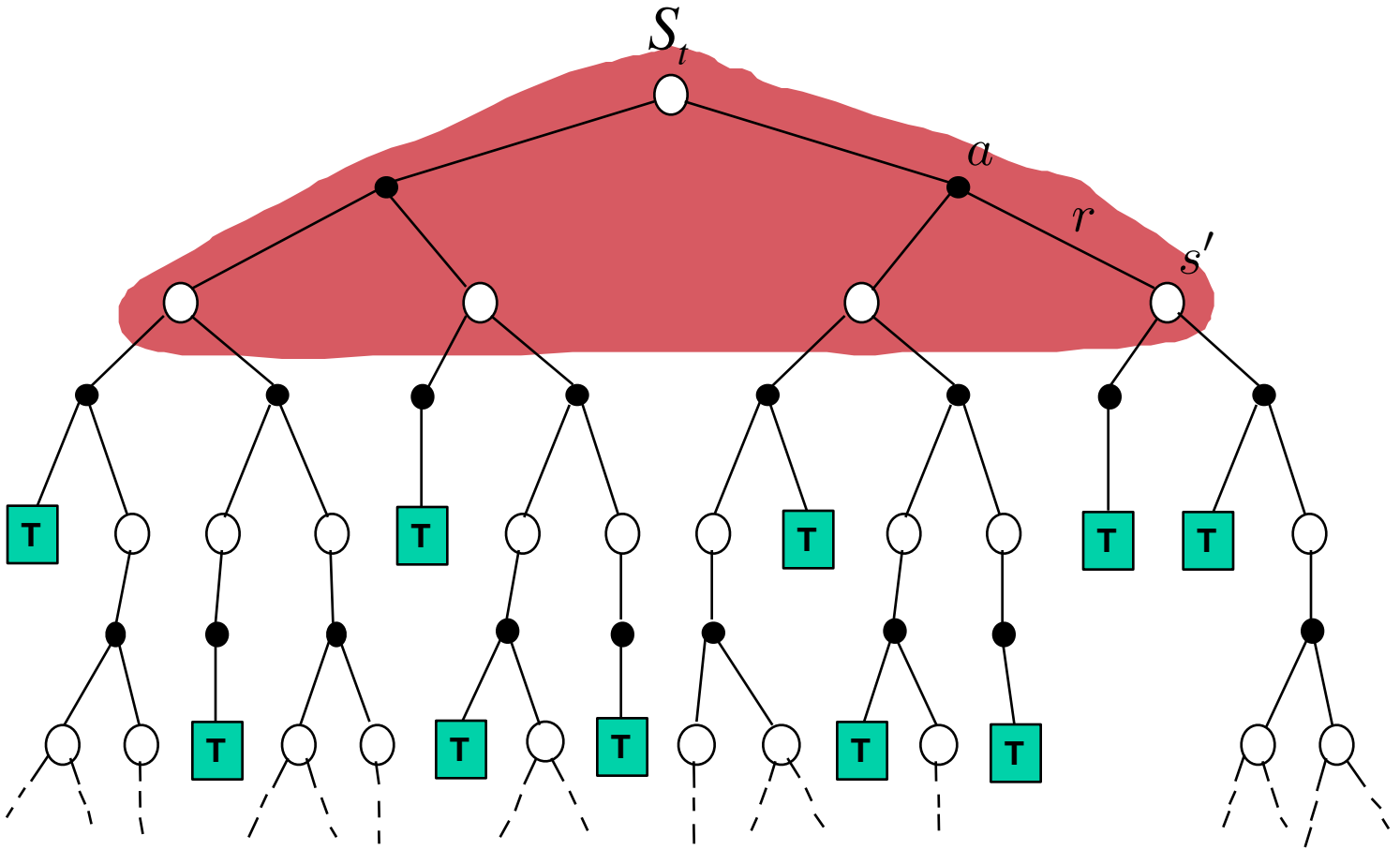
- The **value of an action (in a state)** is the expected return starting after taking that action from that state; depends on the agent's policy:

Action - value function for policy π :

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

Recall: Asynchronous DP Policy Evaluation

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

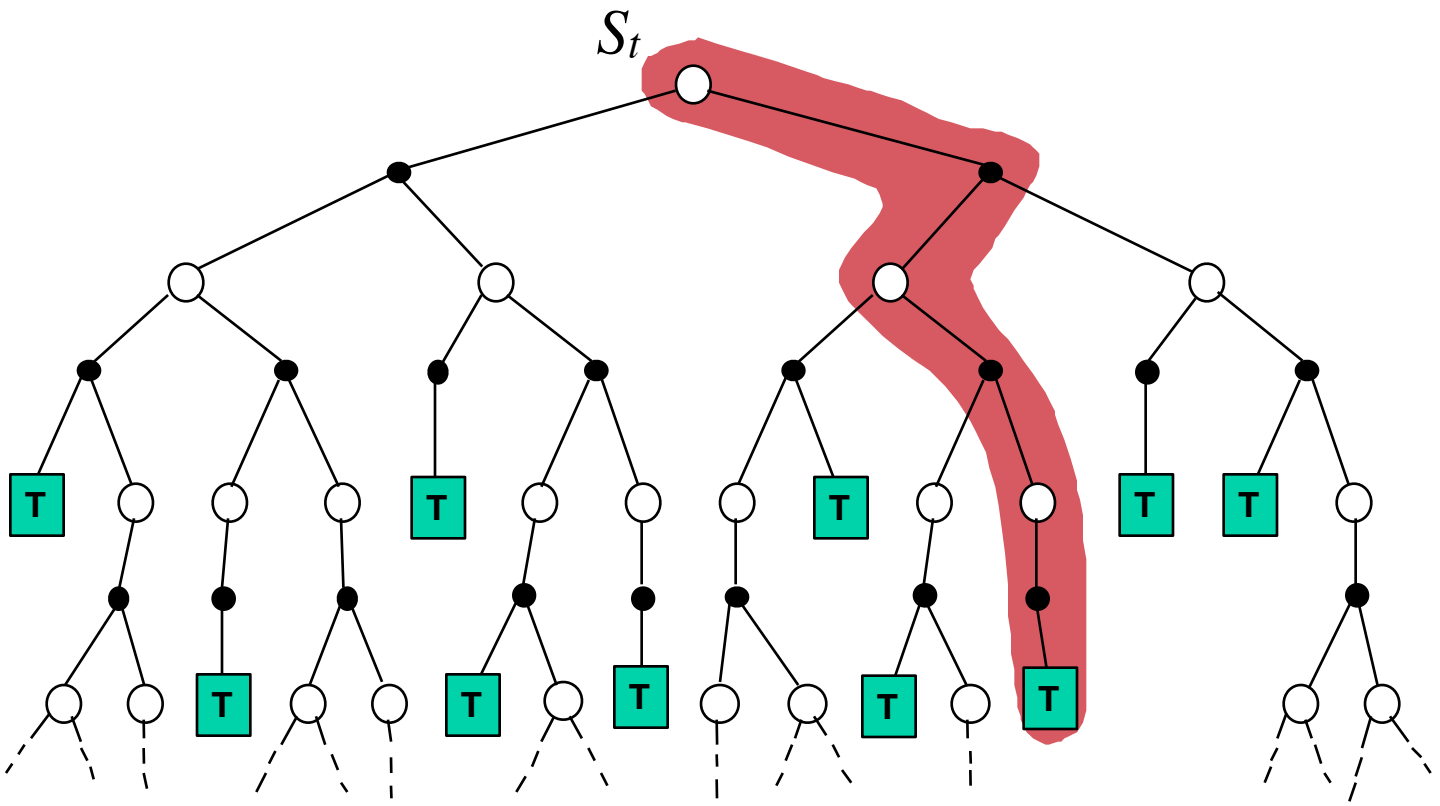


From Planning to Learning

- 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 be hard to learn though
- Direct learning of the value function from interaction
- Still focusing on evaluating a fixed policy

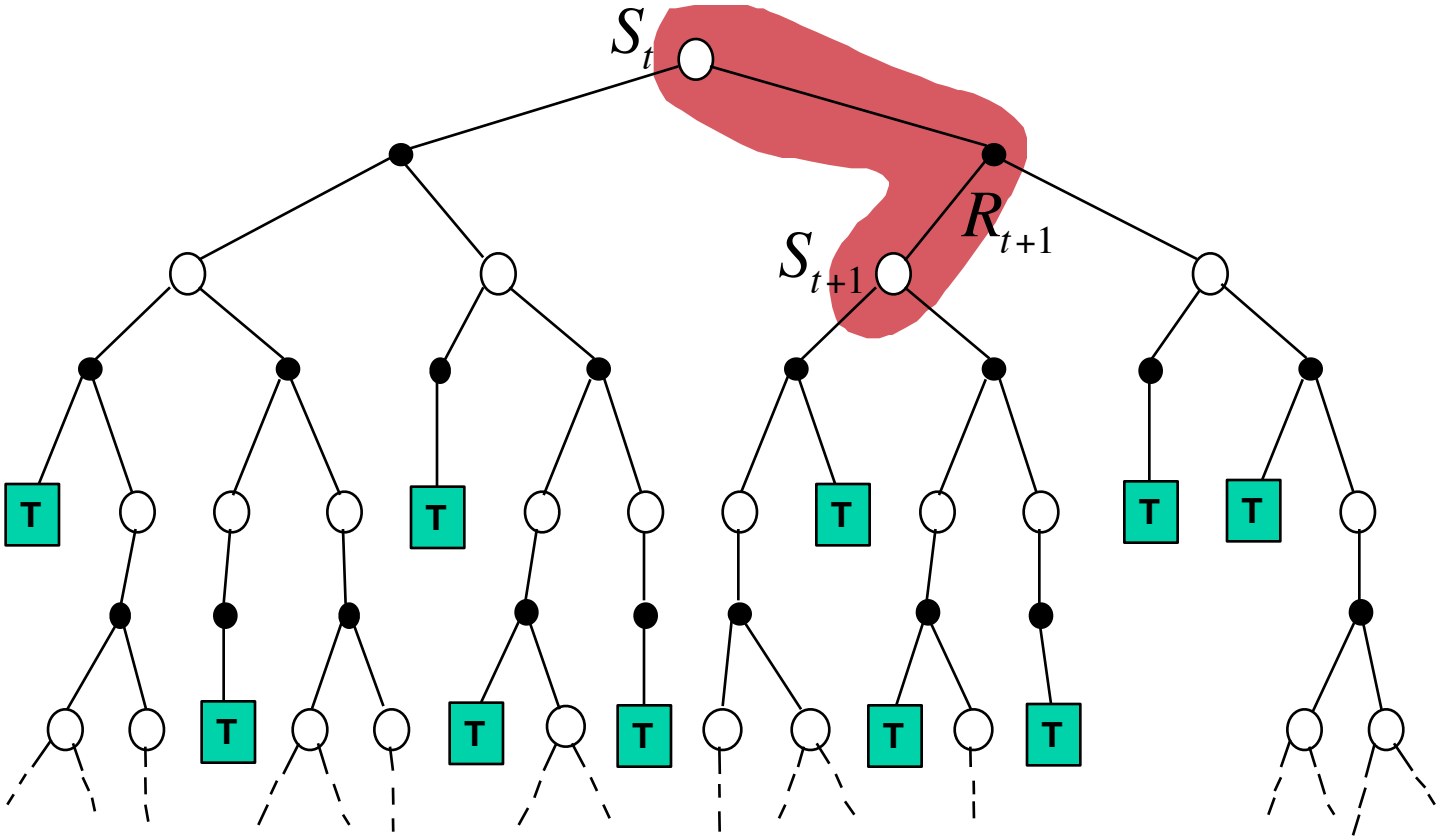
Recall: Simple Monte Carlo

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



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):

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

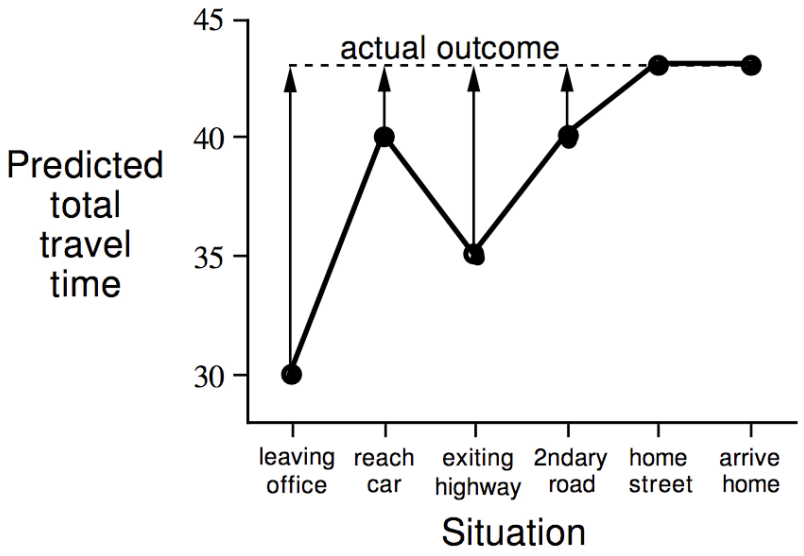
target: an estimate of the return

Example: Driving Home

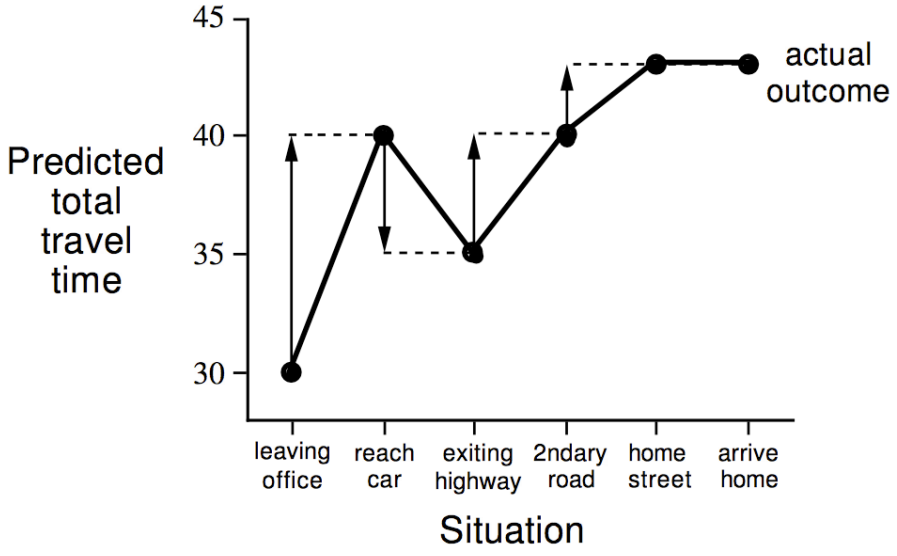
State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)



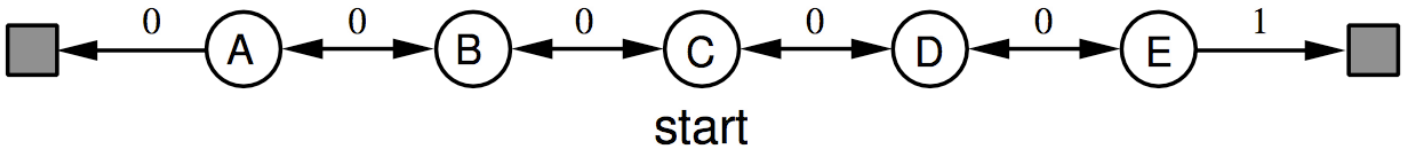
TD methods bootstrap and sample

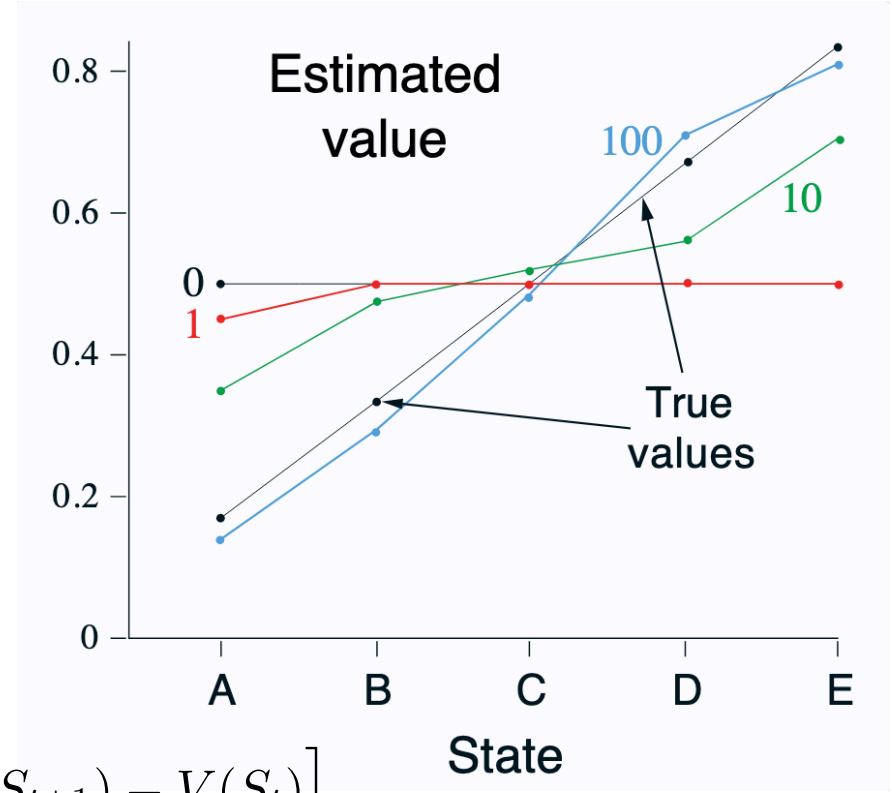
- Bootstrapping: update involves an *estimate*
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an *expected value*
  - MC samples
  - DP does not sample
  - TD samples

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

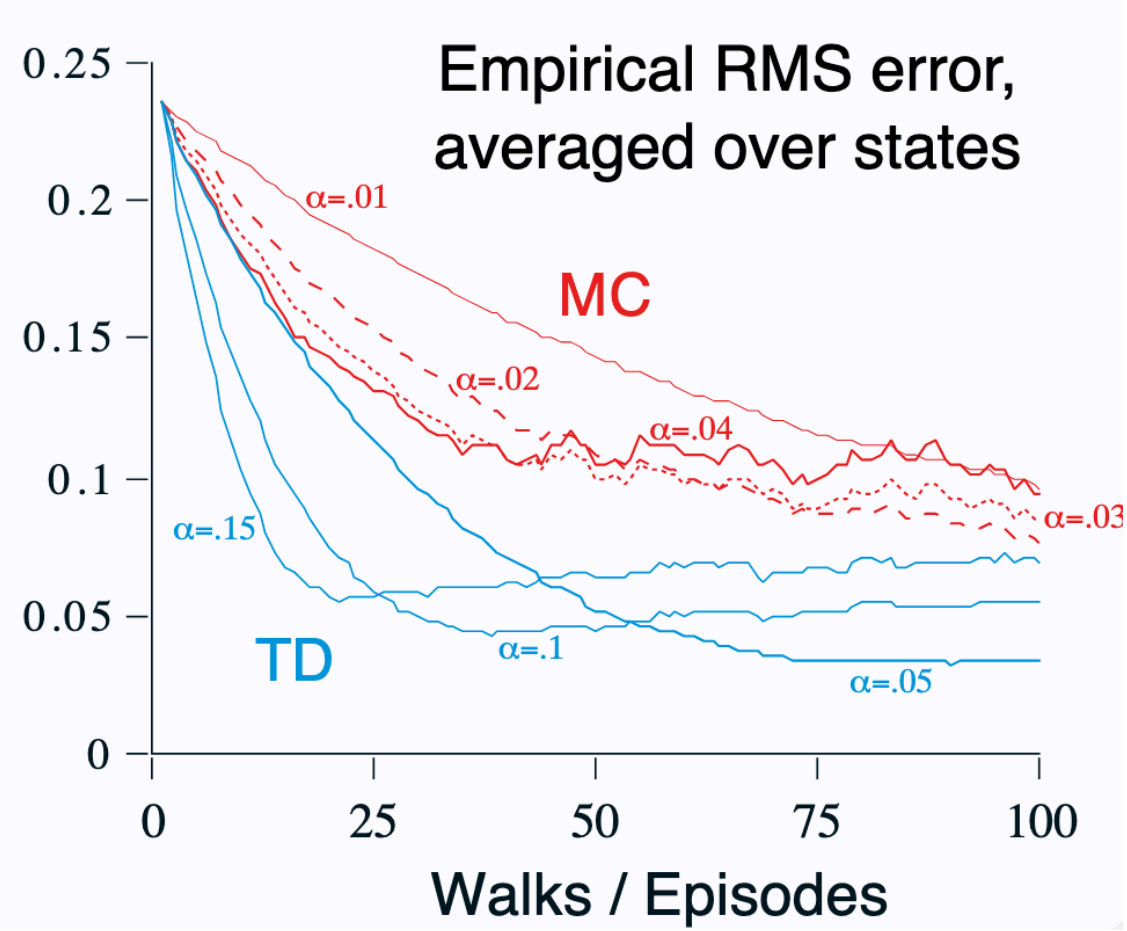




Values learned by TD after various numbers of episodes

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

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Batch Updating in TD and MC methods

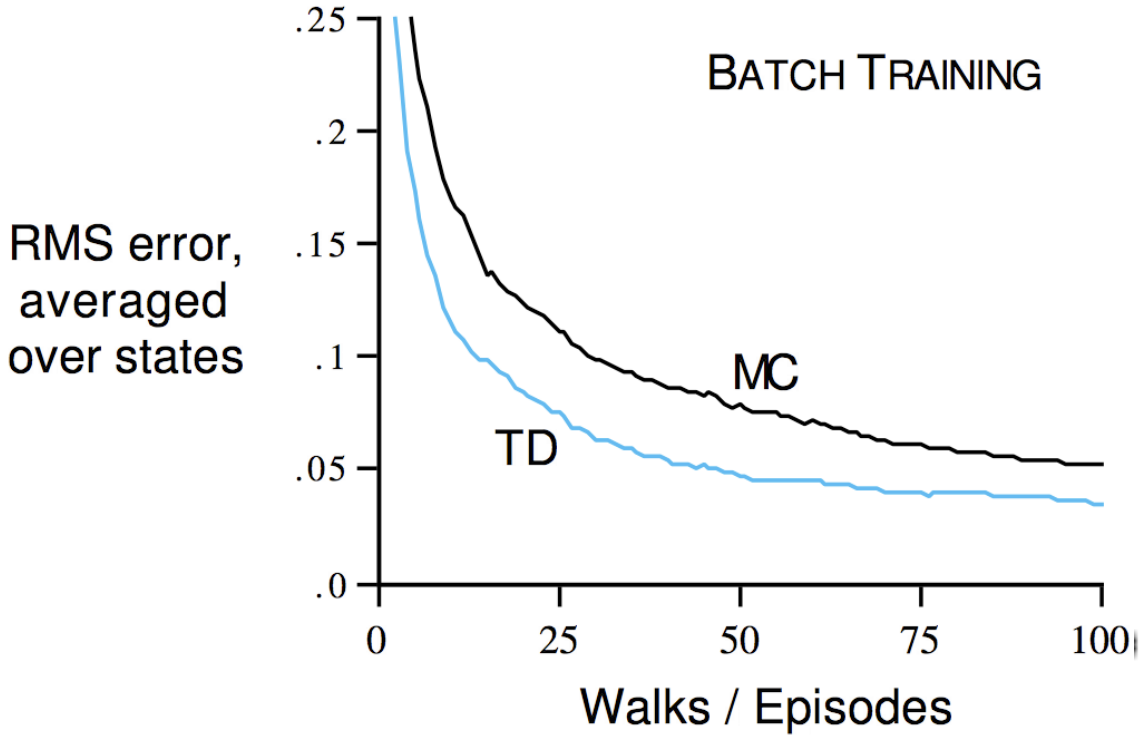
Batch Updating: train completely on a finite amount of data,
e.g., train repeatedly on 10 episodes until convergence.

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:

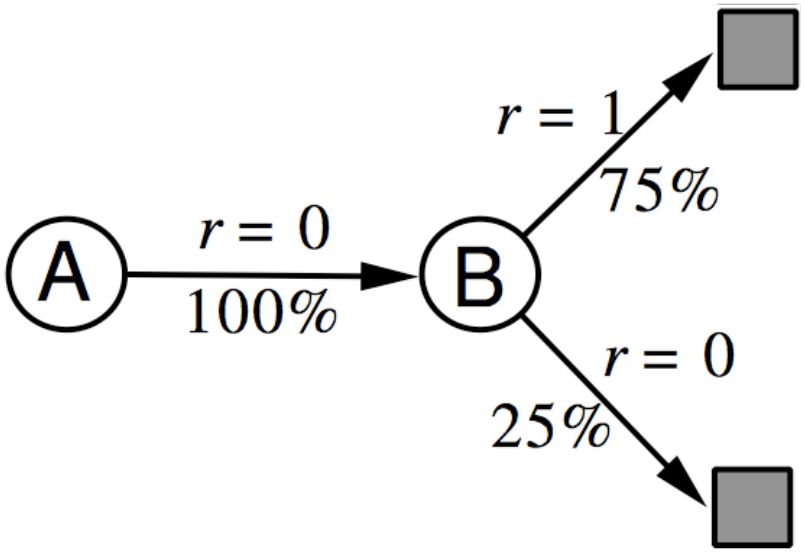
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

$V(\mathrm{B})$?

$V(\mathrm{A})$?

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

You are the Predictor

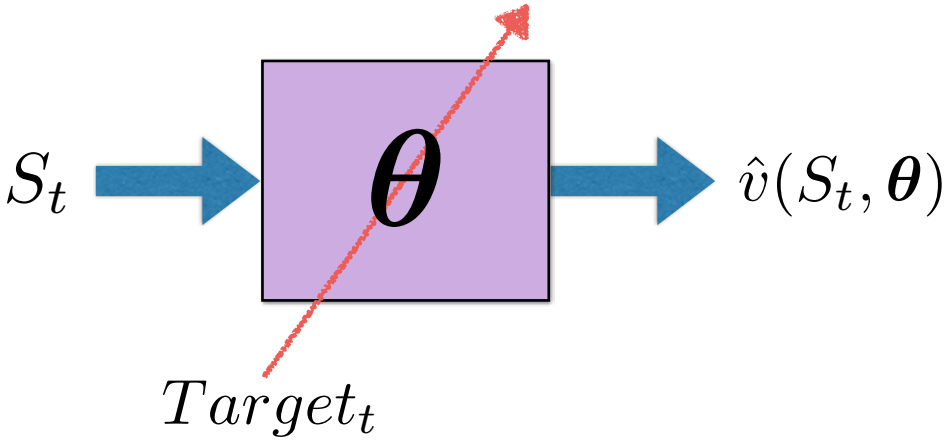


$V(\mathrm{A})?$

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

Recall: Value function approximation



Recall: SGD for learning

General SGD: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \, Error_t^2$

For VFA: $\leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]^2$

Chain rule: $\leftarrow \boldsymbol{\theta} - 2\alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right]$

Semi-gradient: $\leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta})$

Linear case: $\leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t)$

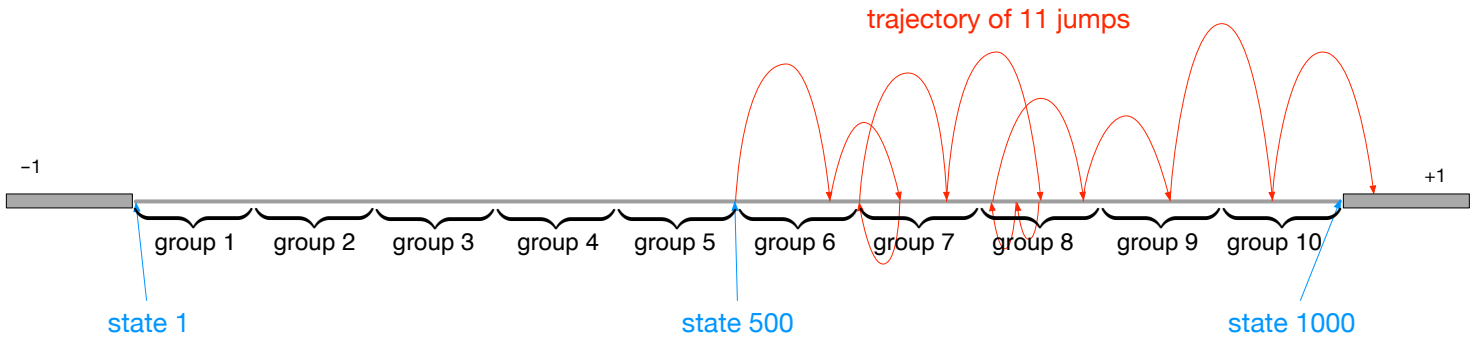
Action-value form: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{q}(S_t, A_t, \boldsymbol{\theta}) \right] \boldsymbol{\phi}(S_t, A_t)$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^n \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\boldsymbol{\theta}$ arbitrarily (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R + \gamma\hat{v}(S',\boldsymbol{\theta}) - \hat{v}(S,\boldsymbol{\theta})\big]\nabla\hat{v}(S,\boldsymbol{\theta})$
        $S \leftarrow S'$
    until S' is terminal

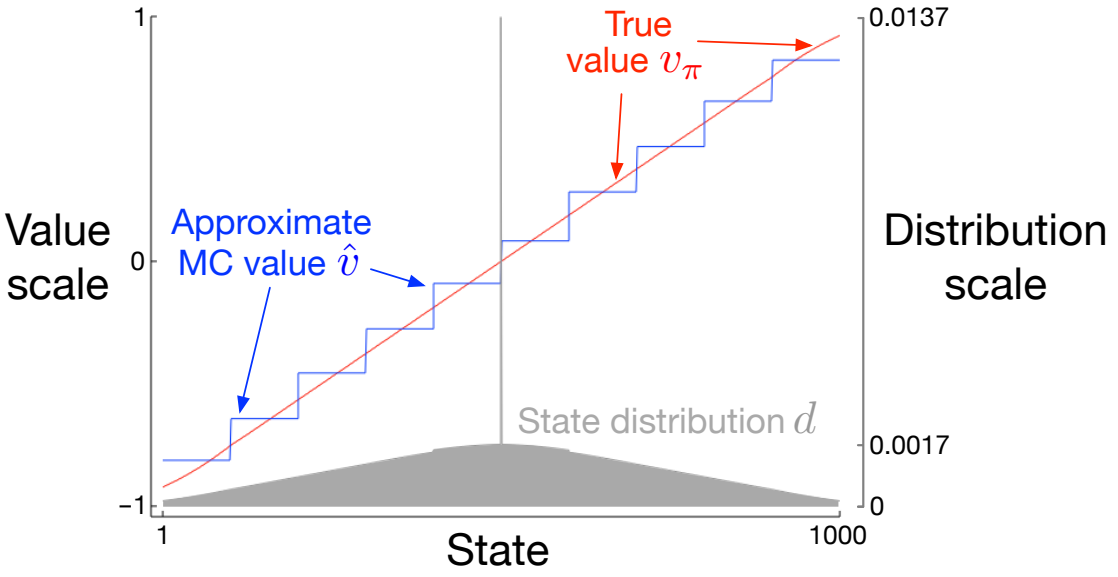
Recall: State aggregation example



The whole value function over 1000 states will be approximated with 10 numbers!

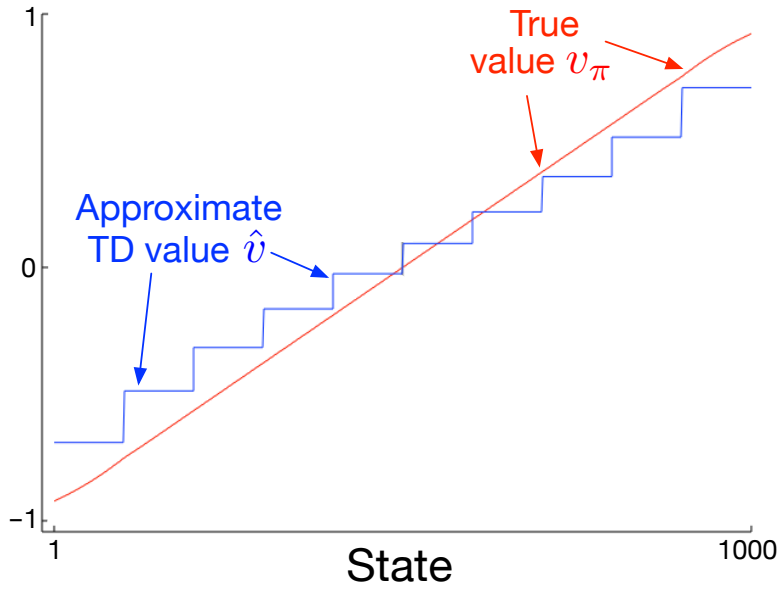
Recall: Gradient MC results

- 10 groups of 100 states
- after 100,000 episodes
- α = 2 x 10^{-5}
- state distribution affects accuracy



Gradient TD is less accurate than MC on the 1000-state random walk using state aggregation

- 10 groups of 100 states
- after 100,000 episodes
- α = 2 x 10^{-5}



Relative values are still pretty accurate

Unified View