Chapter 6: Temporal Difference Learning

Objectives of this chapter:

- 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 :

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

target: the actual return after time t

The simplest TD method, TD(0) :

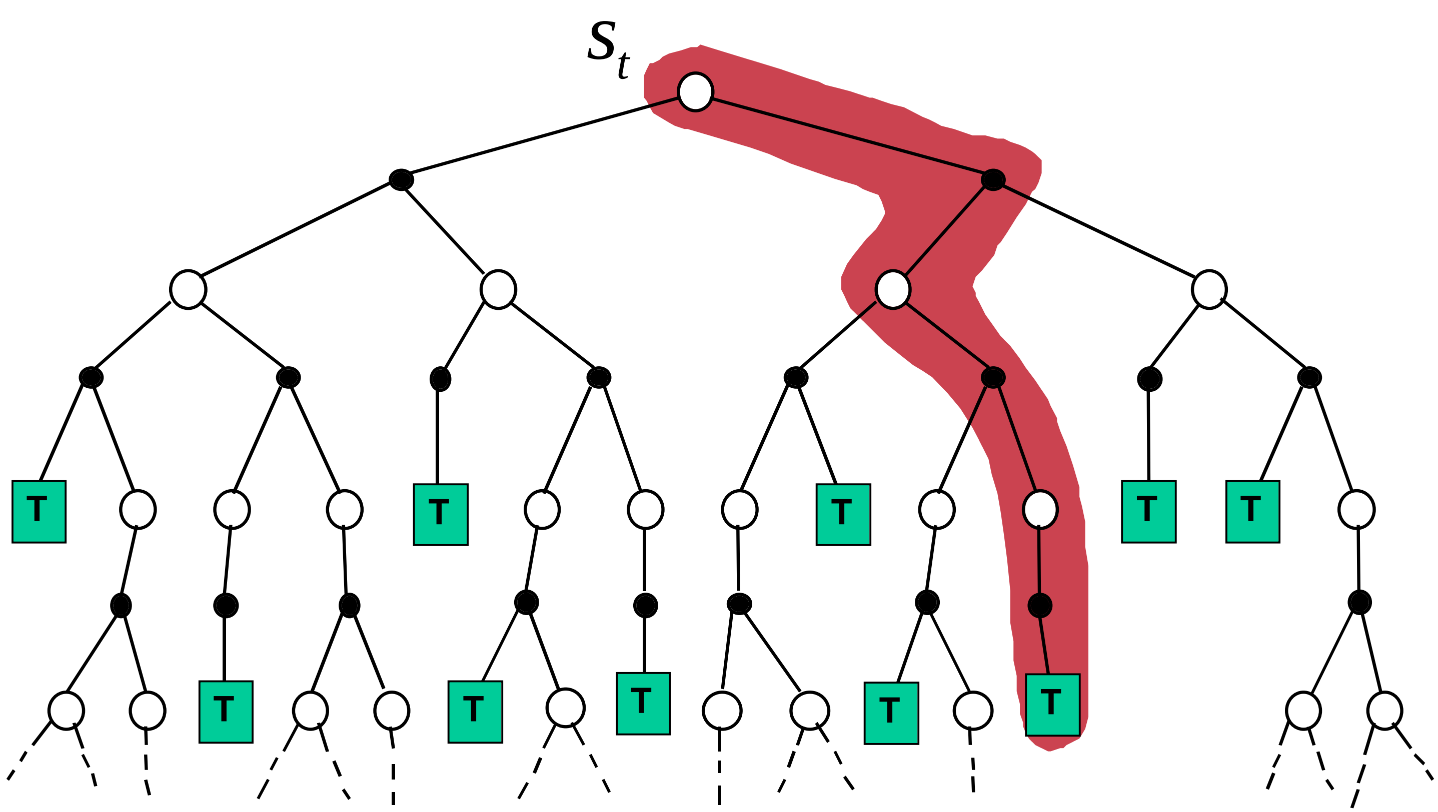
$$V(s_t) \leftarrow V(s_t) + \alpha \left[\underbrace{r_{t+1} + \gamma V(s_{t+1})} - V(s_t) \right]$$

target: an estimate of the return

Simple Monte Carlo

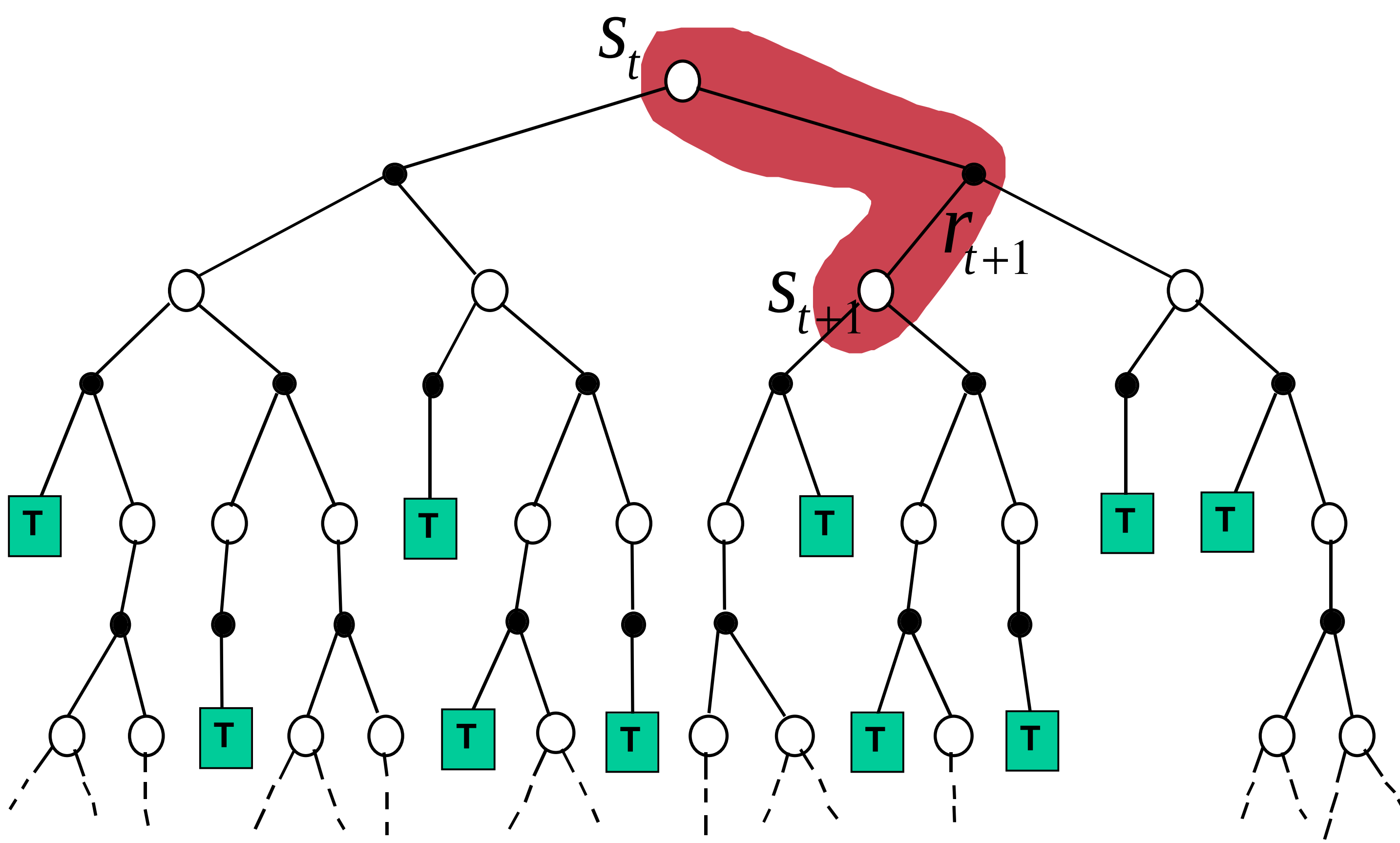
$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



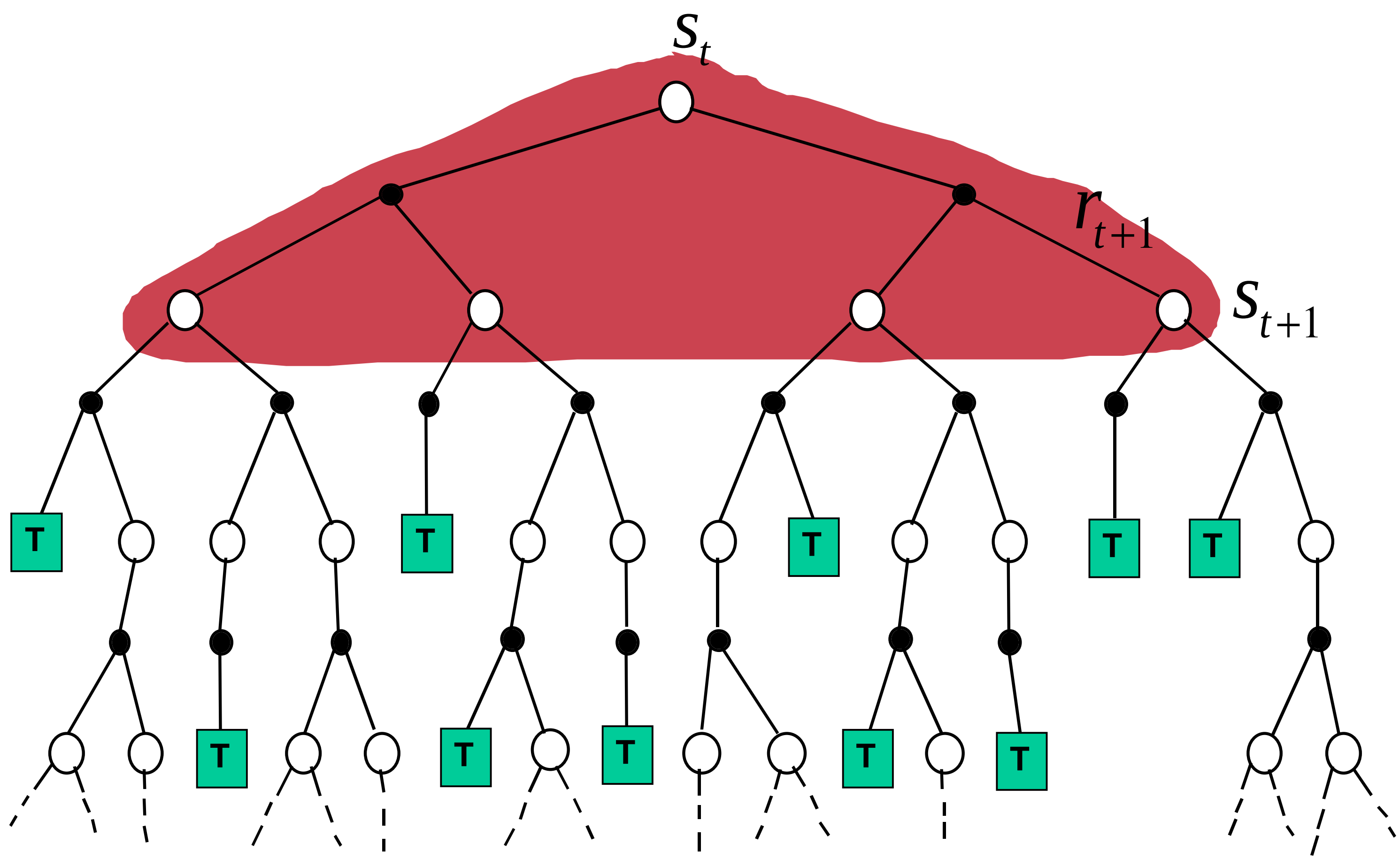
Simplest TD Method

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



cf. Dynamic Programming

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



TD Bootstraps and Samples

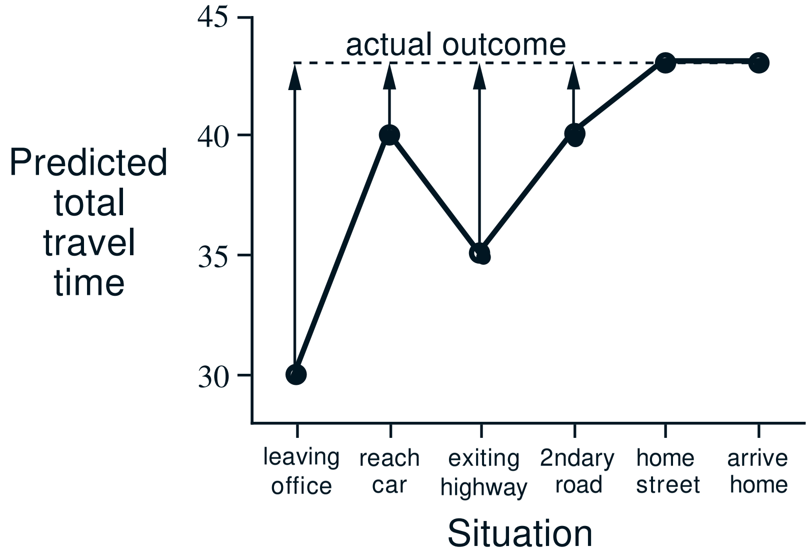
- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an expected value
  - MC samples
  - DP does not sample
  - TD samples

Example: Driving Home

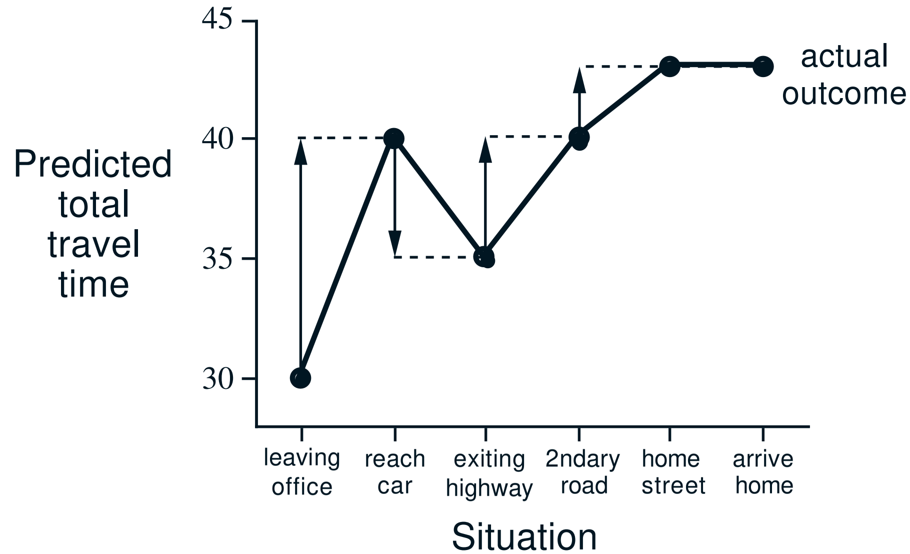
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)



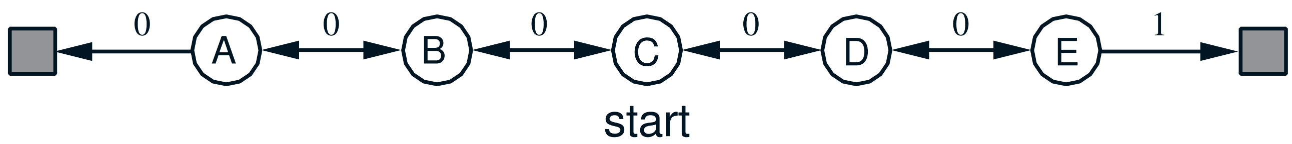
Changes recommended by TD methods (α=1)

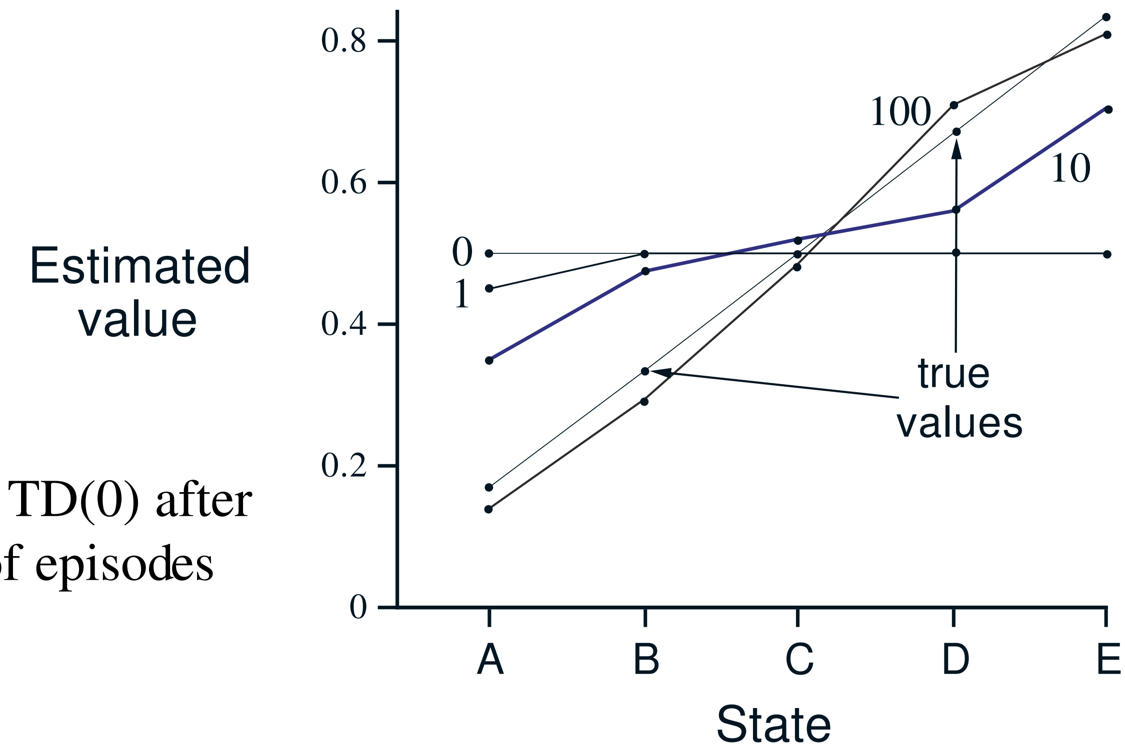


Advantages of TD Learning

- 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?

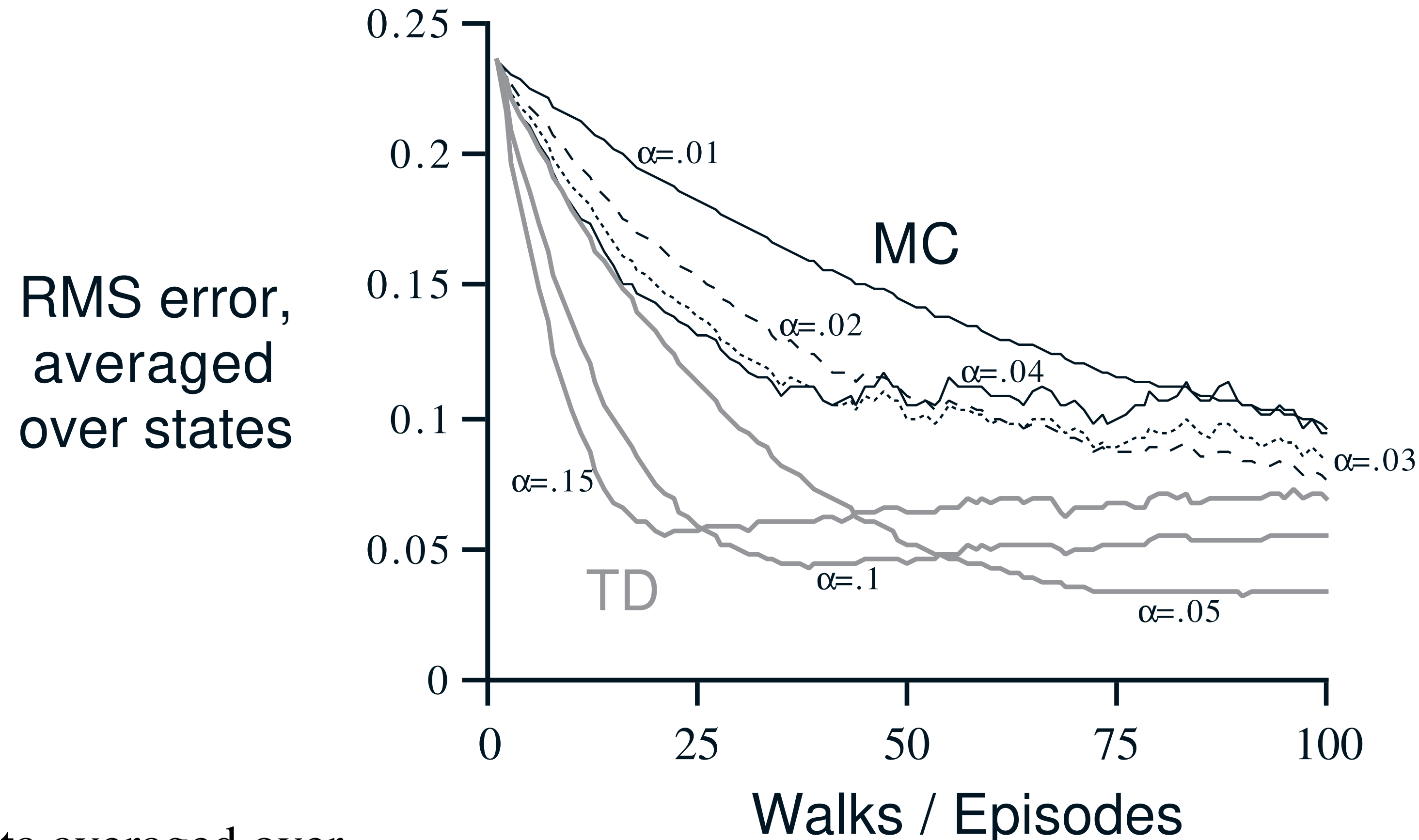
Random Walk Example





Values learned by TD(0) after various numbers of episodes

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Optimality of TD(0)

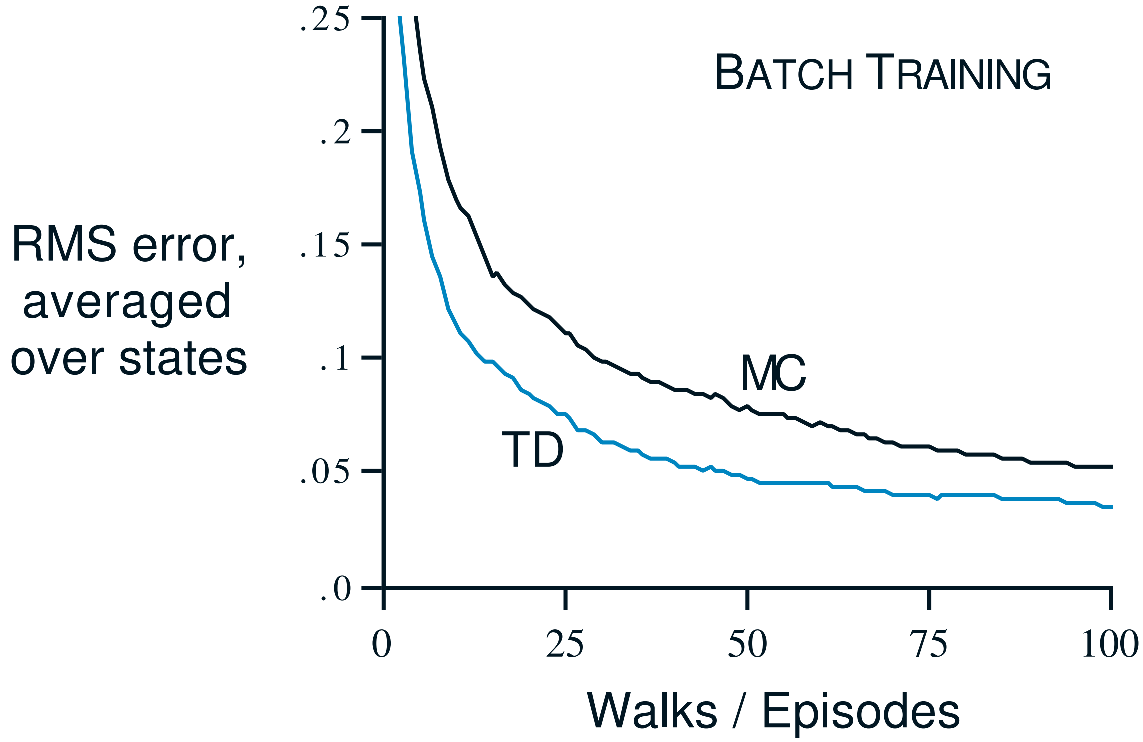
Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.

Compute updates according to TD(0), but only update estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating, TD(0) converges for sufficiently small α.

Constant-α MC also converges under these conditions, but to a difference 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

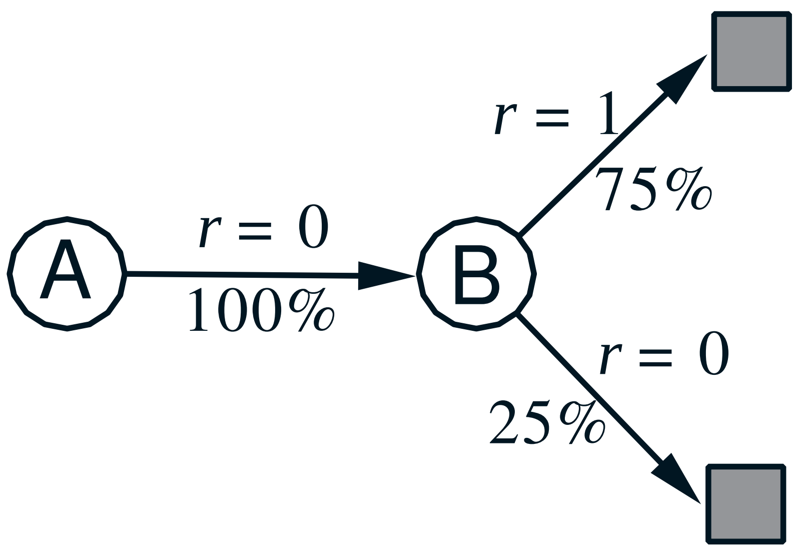
Suppose you observe the following 8 episodes:

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

$V(\mathrm{A})$?

$V(\mathrm{B})$?

You are the Predictor



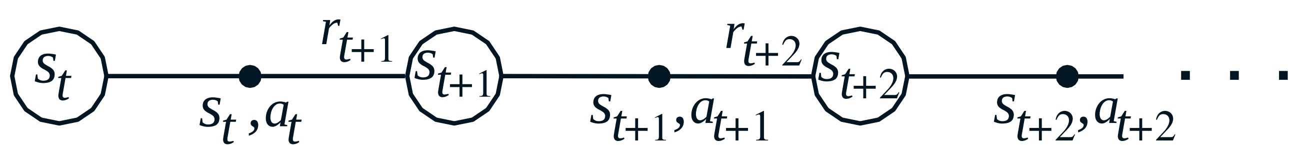
$V(A)$?

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(0) gets

Learning An Action-Value Function

Estimate Q^{π} for the current behavior policy π.



After every transition from a nonterminal state s_t, do this:

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

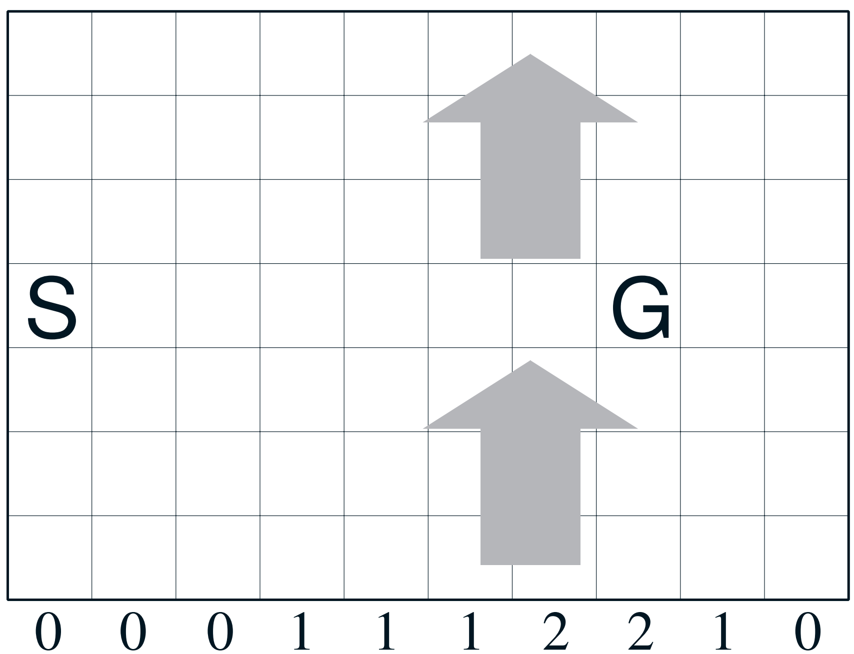
If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

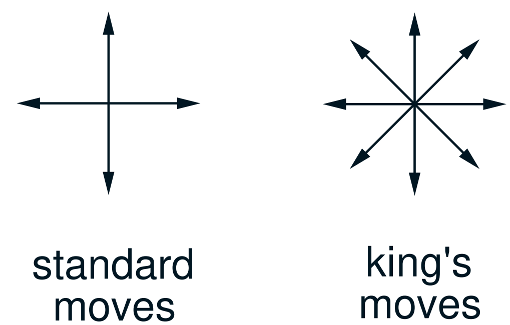
Sarsa: On-Policy TD Control

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

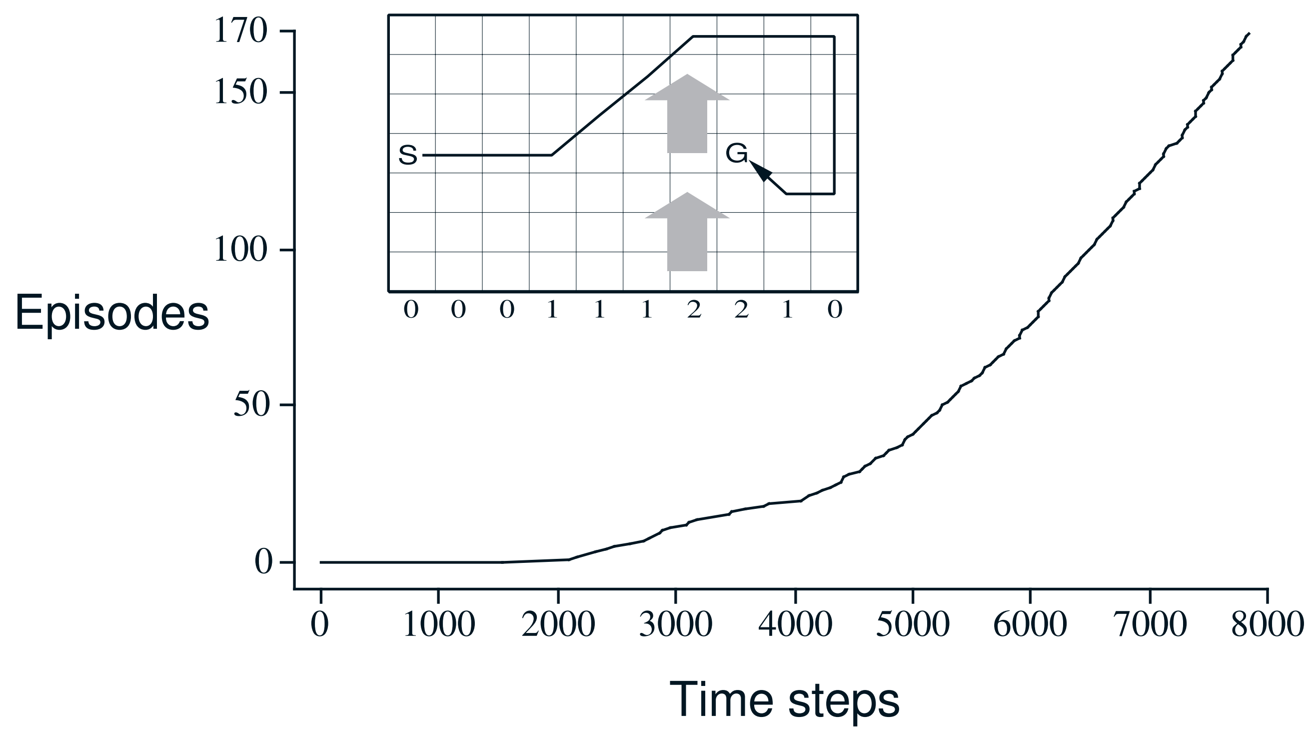
Windy Gridworld





undiscounted, episodic, reward = –1 until goal

Results of Sarsa on the Windy Gridworld



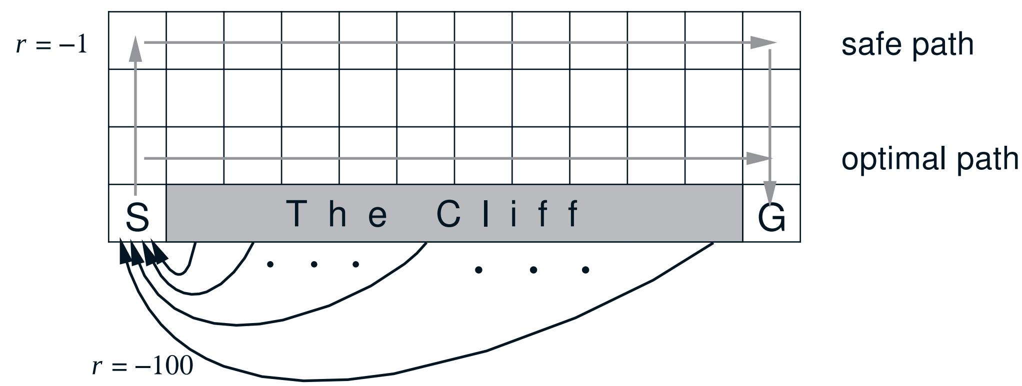
Q-Learning: Off-Ploicy TD Control

One - step Q - learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

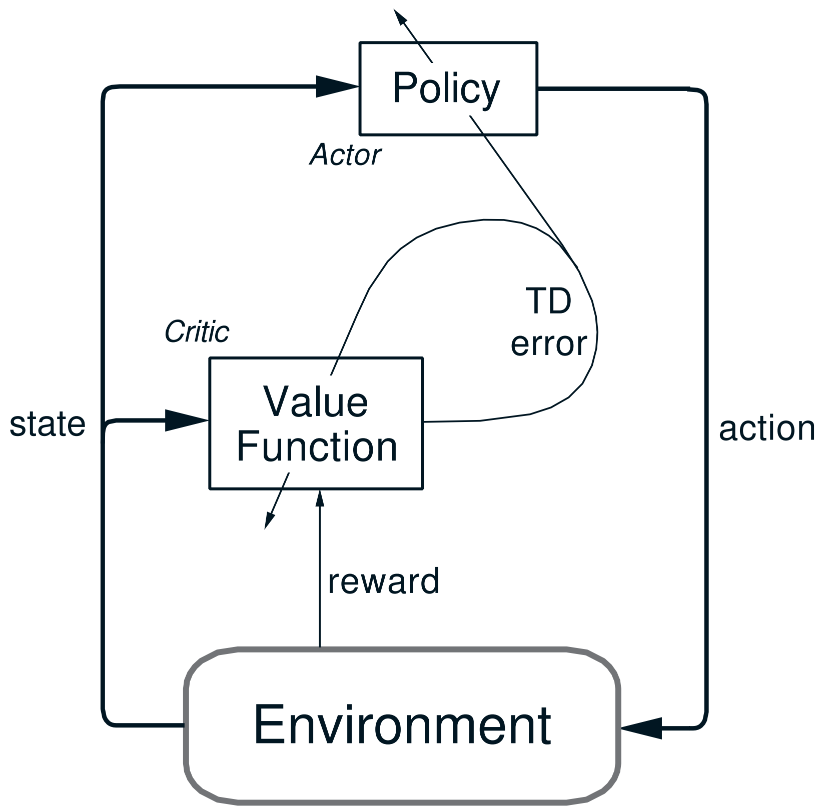
```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ϵ-greedy)
        Take action a, observe r, s'
        Q(s, a) ← Q(s, a) + α [r + γ max_a' Q(s', a') − Q(s, a)]
        s ← s';
    until s is terminal
```

Cliffwalking



ε−greedy, $\varepsilon = 0.1$

Actor-Critic Methods



- Explicit representation of policy as well as value function
- Minimal computation to select actions
- Can learn an explicit stochastic policy
- Can put constraints on policies
- Appealing as psychological and neural models

Actor-Critic Details

TD - error is used to evaluate actions:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

If actions are determined by preferences, $p(s,a)$, as follows:

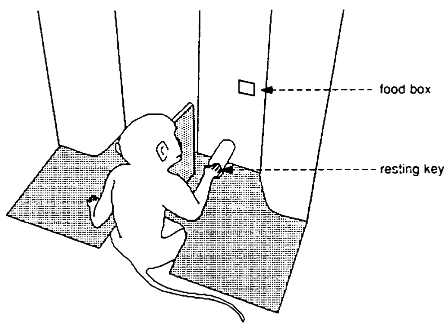
$$\pi_t(s,a) = \Pr\left\{a_t = a \middle| s_t = s\right\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}},$$

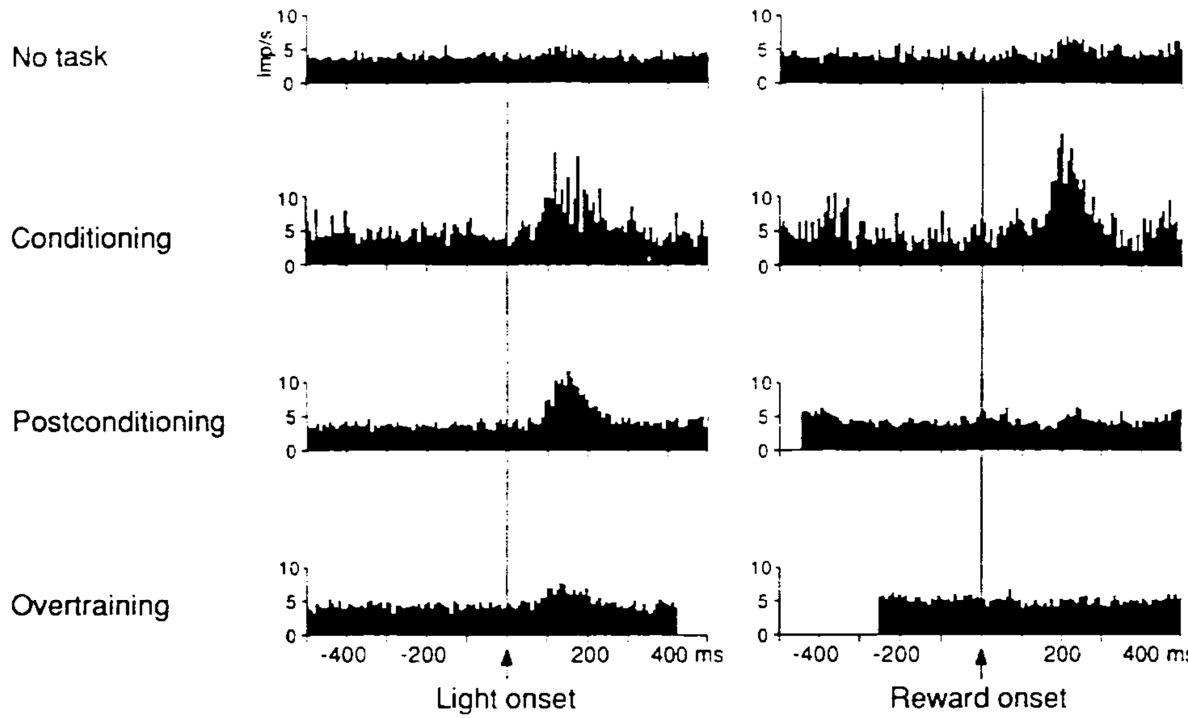
then you can update the preferences like this:

$$p(s_t,a_t) \leftarrow p(s_t,a_t) + \beta\,\delta_t$$

Dopamine Neurons and TD Error

W. Schultz et al.
Universite de Fribourg





Average Reward Per Time Step

Average expected reward per time step under policy π :

$$\rho^{\pi} = \lim_{n \to \infty} \frac{1}{n} \sum_{t=1}^{n} E_{\pi}\{r_t\}$$

the same for each state if ergodic

Value of a state relative to ρ^{π} :

$$\tilde{V}^{\pi}(s) = \sum_{k=1}^{\infty} E_{\pi}\left\{r_{t+k} - \rho^{\pi} \mid s_t = s\right\}$$

Value of a state - action pair relative to ρ^{π} :

$$\tilde{Q}^{\pi}(s,a) = \sum_{k=1}^{\infty} E_{\pi}\left\{r_{t+k} - \rho^{\pi} \mid s_t = s, a_t = a\right\}$$

R-Learning

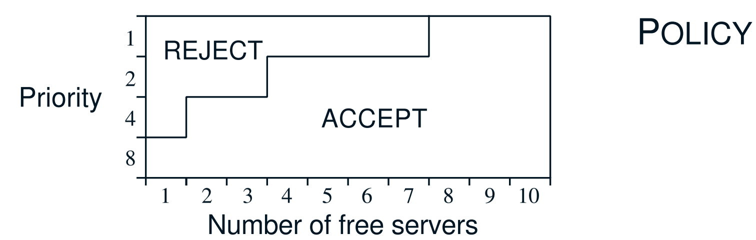
Initialize ρ and $Q(s,a)$, for all s, a, arbitrarily
Repeat forever:
    $s \leftarrow$ current state
    Choose action a in s using behavior policy (e.g., ϵ-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r - \rho + \max_{a'} Q(s',a') - Q(s,a) \right]$
    If $Q(s,a) = \max_a Q(s,a)$, then:
        $\rho \leftarrow \rho + \beta \left[r - \rho + \max_{a'} Q(s',a') - \max_a Q(s,a) \right]$

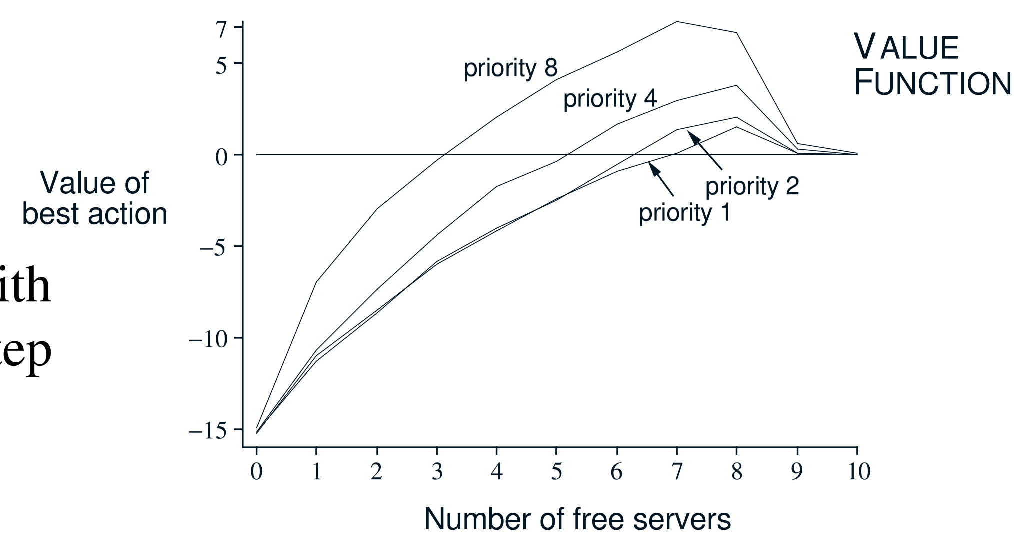
Access-Control Queuing Task

- n servers
- Customers have four different priorities, which pay reward of 1, 2, 3, or 4, if served
- At each time step, customer at head of queue is accepted (assigned to a server) or removed from the queue
- Proportion of randomly distributed high priority customers in queue is h
- Busy server becomes free with probability p on each time step
- Statistics of arrivals and departures are unknown

Apply R-learning

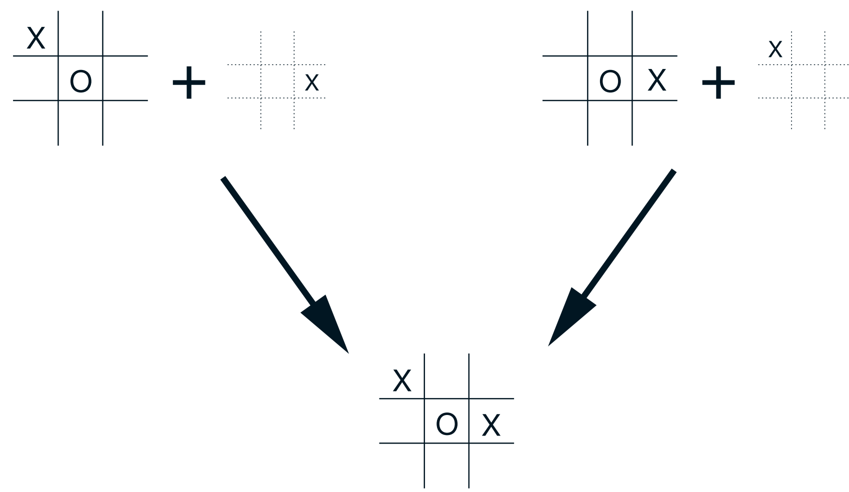
n=10, h=.5, p=.06





Afterstates

- Usually, a state-value function evaluates states in which the agent can take an action.
- But sometimes it is useful to evaluate states after agent has acted, as in tic-tac-toe.
- Why is this useful?



- What is this in general?

Summary

- TD prediction
- Introduced *one-step tabular model-free TD methods*
- Extend prediction to control by employing some form of GPI
  - On-policy control: Sarsa
  - Off-policy control: Q-learning and R-learning
- These methods bootstrap and sample, combining aspects of DP and MC methods