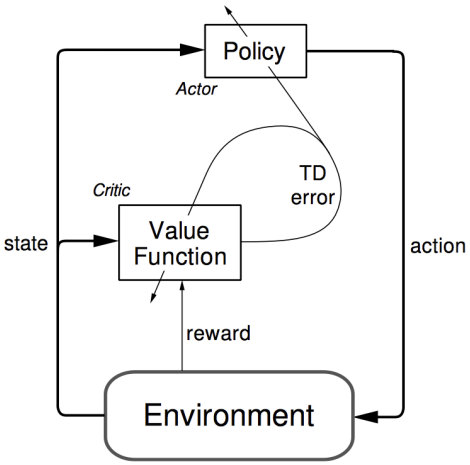
Temporal-Difference Learning

Suggested reading:

Chapter 6 in R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction
MIT Press, 1998.

Temporal-Difference Learning



Contents:

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):

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

target: an estimate of the return

TD Prediction

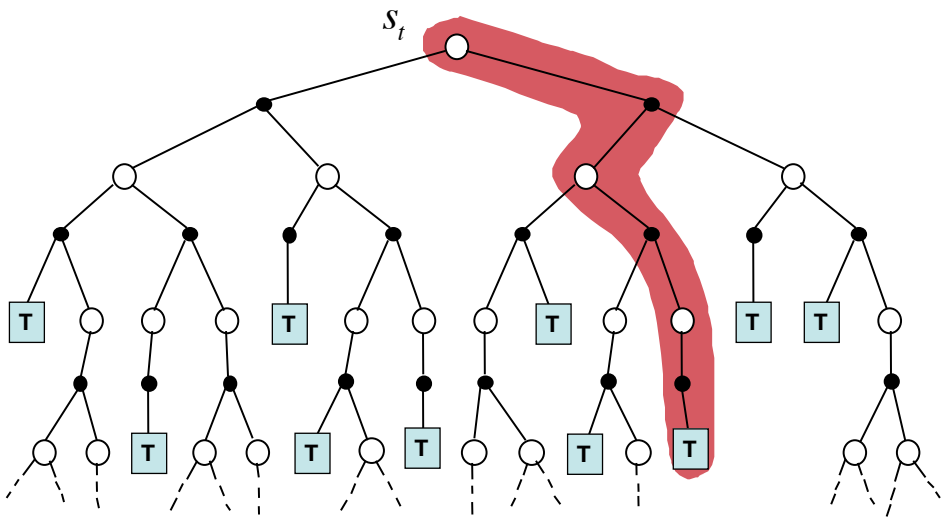
MC, TD and DP make different estimates of the value function:

$$\begin{aligned}
V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} && \text{MC}\\
&= E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\}\\
&= E_\pi\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_t = s\right\}\\
&= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} && \text{TD(0)}\\
&= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^\pi(s')\right] && \text{DP}
\end{aligned}$$

Simple Monte Carlo

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

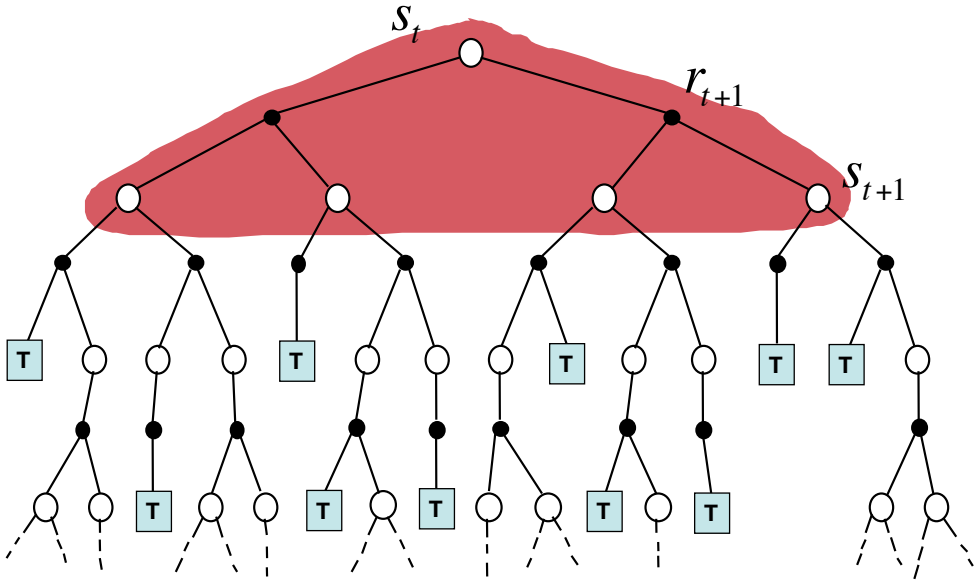
where R_t is the actual return following state s_t.



Monte Carlo uses an estimate of the actual return.

Dynamic Programming

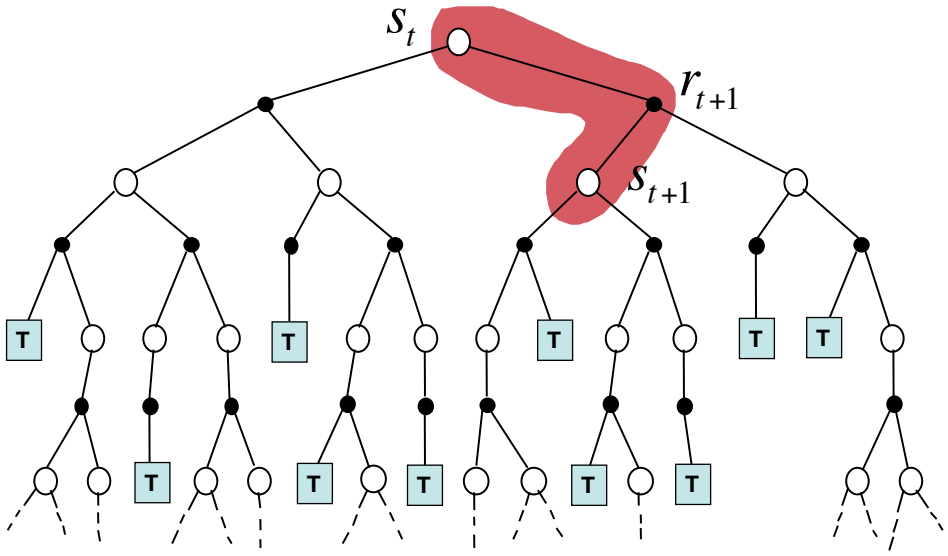
$$V(s_t) \leftarrow E_\pi\left\{r_{t+1} + \gamma V(s_t)\right\}$$



The DP target is an estimate not because of the expected values, which are assumed to be completely provided by a model of the environment, but because V^π is not known and the current estimate is used instead.

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 samples the expected value and uses the current estimate of the value.

TD(0) - Policy-Evaluation

Tabular TD(0) for estimating V^π

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated

Repeat (for each episode):

  Initialize s

  Repeat (for each step of the episode):

    $a \leftarrow$ action given by π for s

    Take action a; observe reward, r, and next state, s'

    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$

    $s \leftarrow s'$

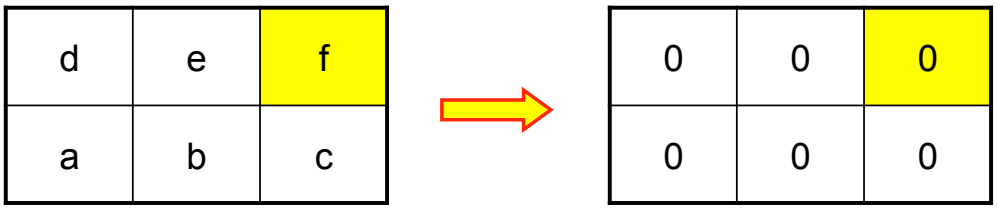
  until s is terminal

„temporal Difference“

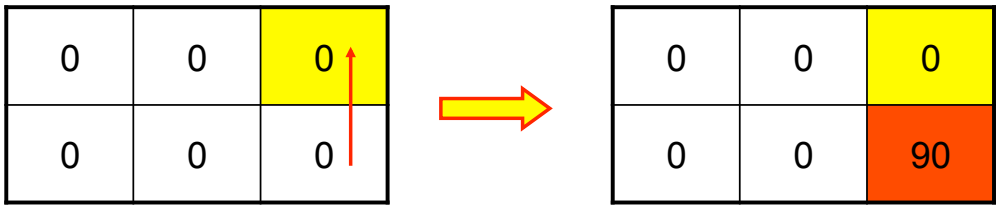
Example

$\alpha = 0.9 \quad \gamma = 1 \quad \pi$ - random

Initialize

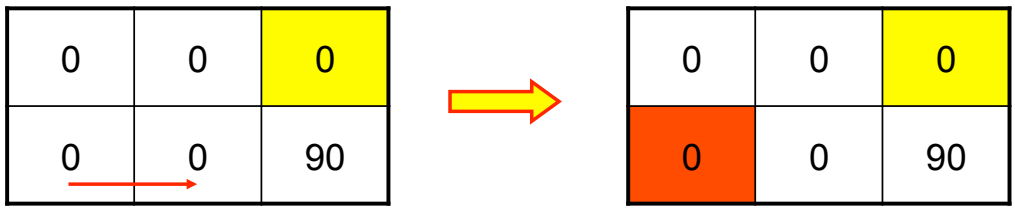


$c \rightarrow f \qquad V(c) \leftarrow 0 + 0.9[100 + 0 - 0] = 90$

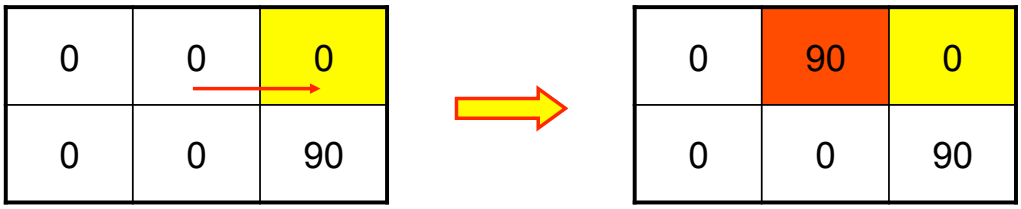


Example

$a \rightarrow b \quad V(a) \leftarrow 0 + 0.9[0 + 0 - 0] = 0$

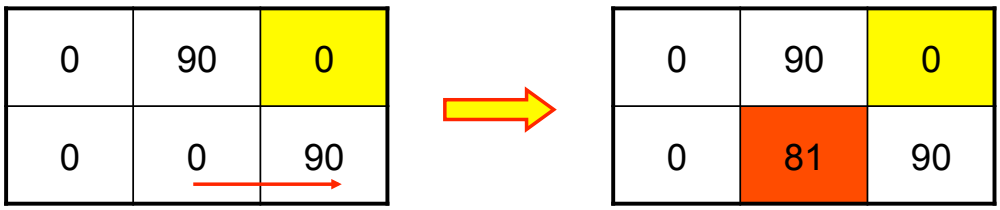


$e \rightarrow f \qquad V(e) \leftarrow 0 + 0.9[100 + 0 - 0] = 90$

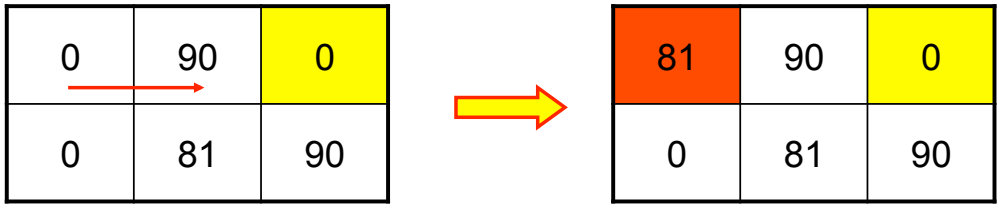


Example

$b \rightarrow c \quad V(b) \leftarrow 0 + 0.9[0 + 90 - 0] = 81$



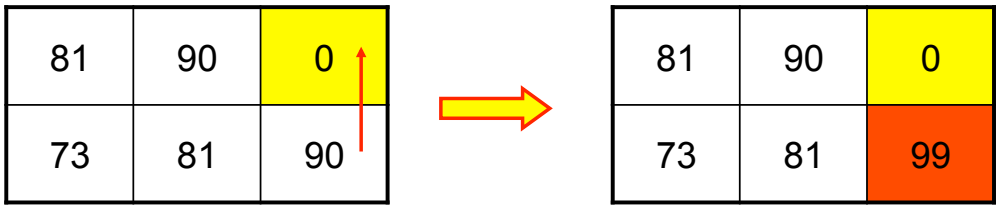
$d \rightarrow e \quad V(d) \leftarrow 0 + 0.9[0 + 90 - 0] = 81$



Example

$a \rightarrow b \quad V(a) \leftarrow 0 + 0.9[0 + 81 - 0] \approx 73$

$c \rightarrow f \quad V(c) \leftarrow 90 + 0.9[100 + 0 - 90] = 99$



Example

$e \rightarrow f \quad V(e) \leftarrow 90 + 0.9[100 + 0 - 90] = 99$

81	90	0
73	81	99

81	99	0
73	81	99

$c \rightarrow b \quad V(c) \leftarrow 99 + 0.9[0 + 81 - 99] \approx 83$

81	99	0
73	81	99

81	99	0
73	81	83

Example

$\gamma = 1$

100	100	0
100	100	100

$\gamma = 0.9$

52	66	0
49	57	76

TD methods bootstrap and sample

- Bootstrapping: update uses an *estimate* of the successor state
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update looks ahead for a sample successor state
  - MC samples
  - DP does not sample
  - TD samples

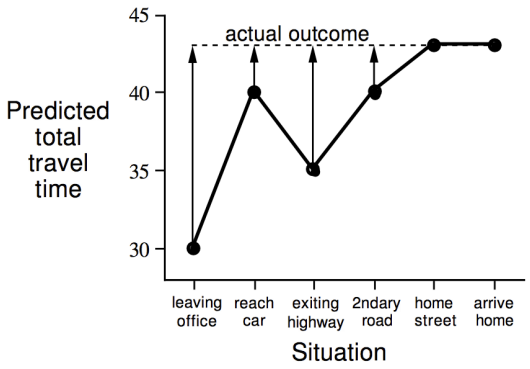
Example: Driving Home

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

- The “rewards” are the elapsed times on each leg of the journey.
- We are not discounting ($\gamma=1$), and thus the return for each state is the actual time to go from that state.
- The value of each state is the *expected time to go.*

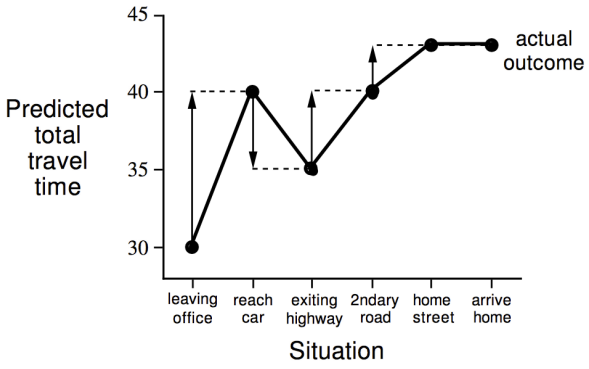
Driving Home

Changes recommended by Monte Carlo methods (α=1)



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

Changes recommended by TD methods (α=1)



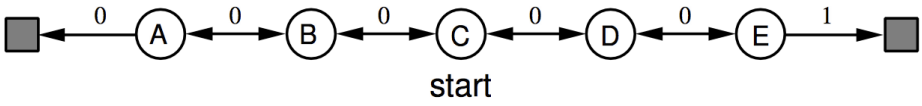
$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

MC must wait with the update until the final outcome!

Advantages of TD Prediction Methods

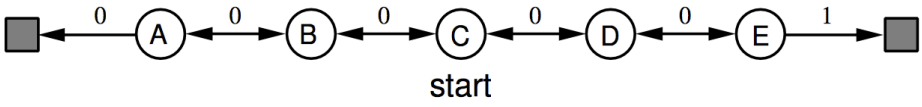
- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Random Walk Example

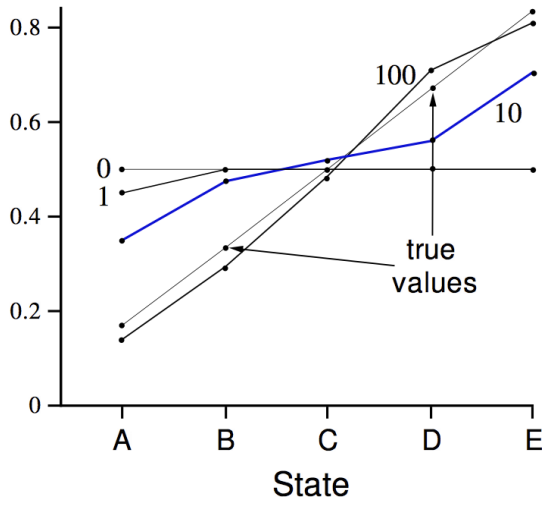


- All episodes start in the center state C
- proceed either left or right by one state on each step, with equal probability
- Episodes terminate either on the extreme left or the extreme right.
- When an episode terminates on the right a reward of 1 occurs; all other rewards are zero.
- Because this task is undiscounted and episodic, the true value of each state is the probability of terminating on the right if starting from that state.
- The true values of all the states, A through E, are 1/6, 2/6, 3/6, 4/6, 5/6

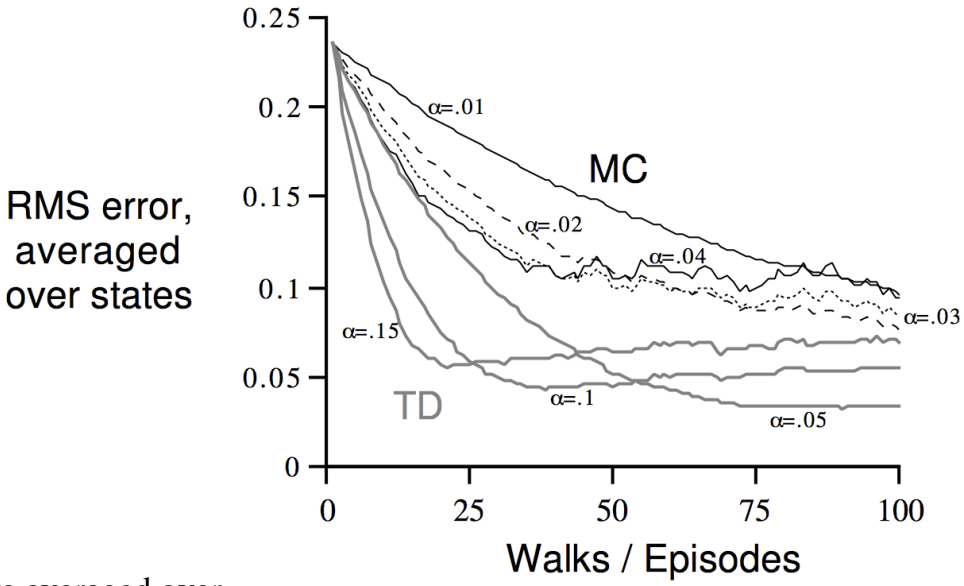
Random Walk Example



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)

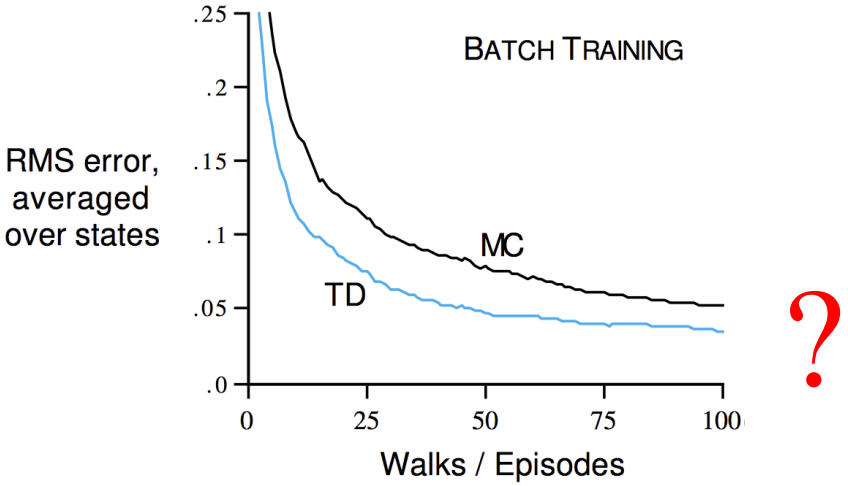
Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.

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.

TD vs. MC

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

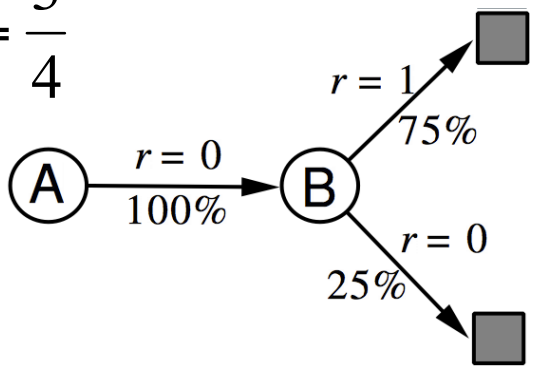
This means that the first episode started in state *A*, transitioned to *B* with a reward of 0, and then terminated from *B* with a reward of 0.

The other seven episodes were even shorter, starting from *B* and terminating immediately.

What is the optimal value for the estimate V(*A*) given this data?

TD vs. MC

$$V(B) = \frac{3}{4}$$



$$V(\mathrm{A})? = \begin{cases} \frac{3}{4} \\ 0 \end{cases}$$

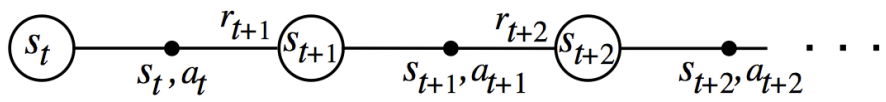
TD vs. MC

- 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
  - This is called the certainty-equivalence estimate, because it is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximated.
  - This is what TD(0) gets

Sarsa: On-Policy TD Control

Sarsa: Learning An Action-Value Function

Estimate Q^π for the current behavior policy π.



After every transition from a nonterminal state s_t, do:

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

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:

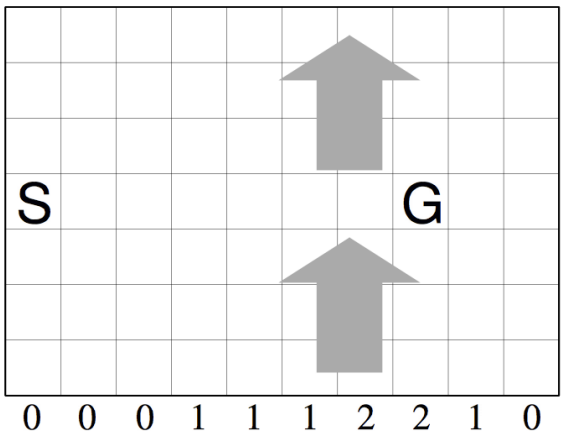
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a)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

On-Policy

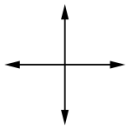
$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ Quintupel

Sarsa: On-Policy TD Control

Move from S to G, but consider the crosswind that moves you upward. For example, if you are one cell to the right of the goal, then the action left takes you to the cell just above the goal.



Windy Gridworld

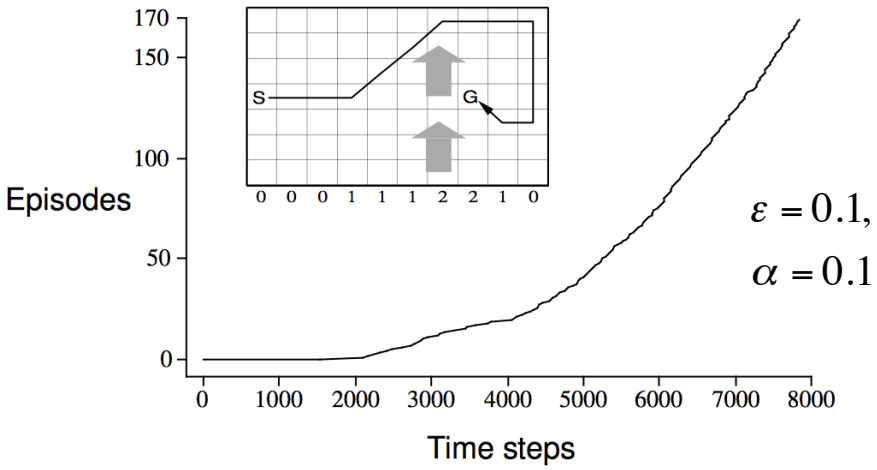


standard moves

strength of the wind

undiscounted, episodic task with constant rewards
reward = –1 until goal

Sarsa: On-Policy TD Control



Results of Sarsa on the Windy Gridworld

Can Monte Carlo methods be used on this task?

No, since termination is not guaranteed for all policies.

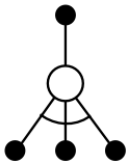
And Sarsa?

Step-by-step learning methods (e.g. Sarsa) do not have this problem. They quickly learn during the episode that such policies are poor, and switch to something else.

Q-Learning: Off-Policy 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'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
  until s is terminal

Off-Policy

Example

d	e	f
a	b	c

$\alpha = 0.9 \quad \gamma = 1$

Initialize $Q(s,a) = 0$

d	e	f
a	b	c

$c \rightarrow f$

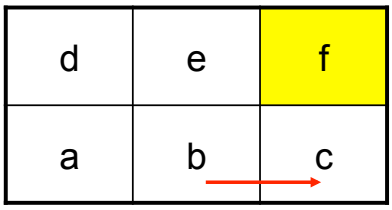
$Q(c,\uparrow) \leftarrow 0 + 0.9[100 + 0 - 0] = 90$

d	e	f
a	b	c

$e \rightarrow f$

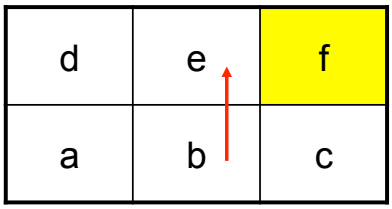
$Q(e,\rightarrow) \leftarrow 0 + 0.9[100 + 0 - 0] = 90$

Example



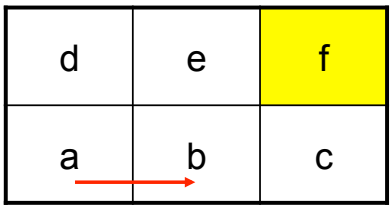
$b \rightarrow c$

$Q(b,\rightarrow) \leftarrow 0 + 0.9[0 + 90 - 0] = 81$



$b \rightarrow e$

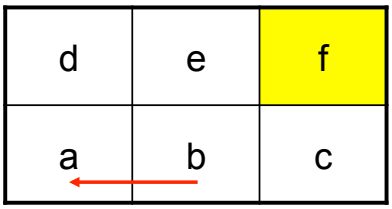
$Q(b,\uparrow) \leftarrow 0 + 0.9[0 + 90 - 0] = 81$



$a \rightarrow b$

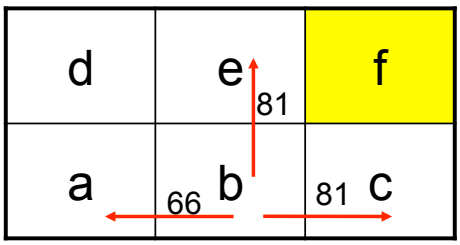
$Q(a,\rightarrow) \leftarrow 0 + 0.9[0 + 81 - 0] \approx 73$

Example

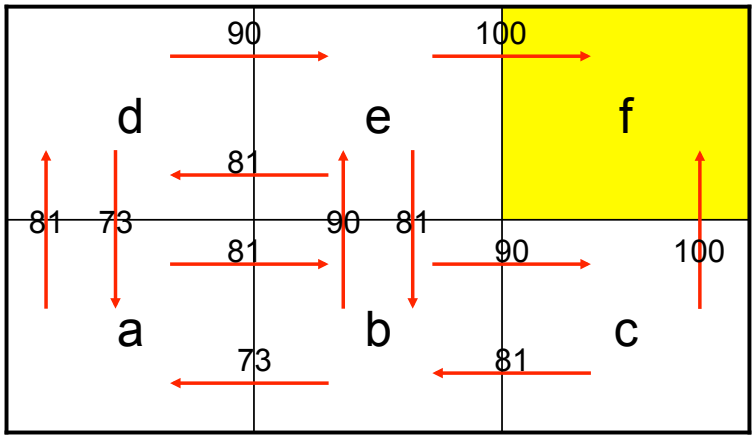


$b \rightarrow a$

$Q(b,\leftarrow) \leftarrow 0 + 0.9[0 + 73 - 0] \approx 66$

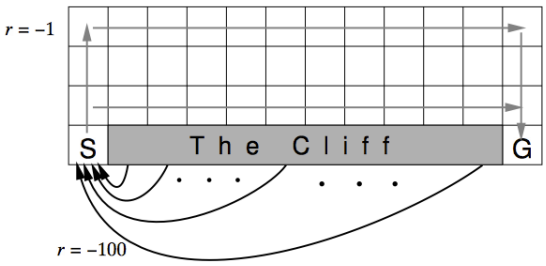


Example

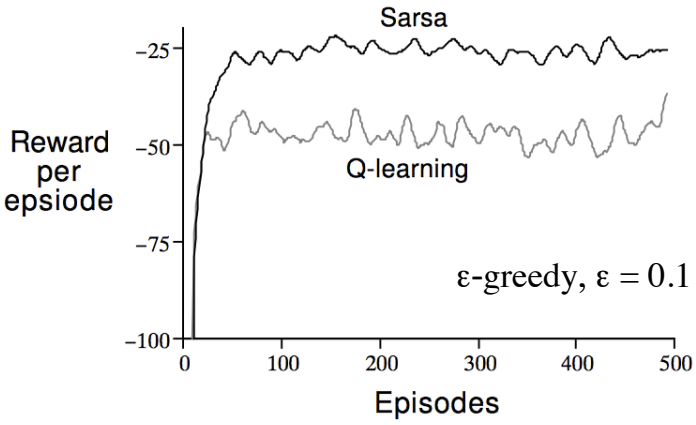


$\gamma = 0.9$

Cliffwalking



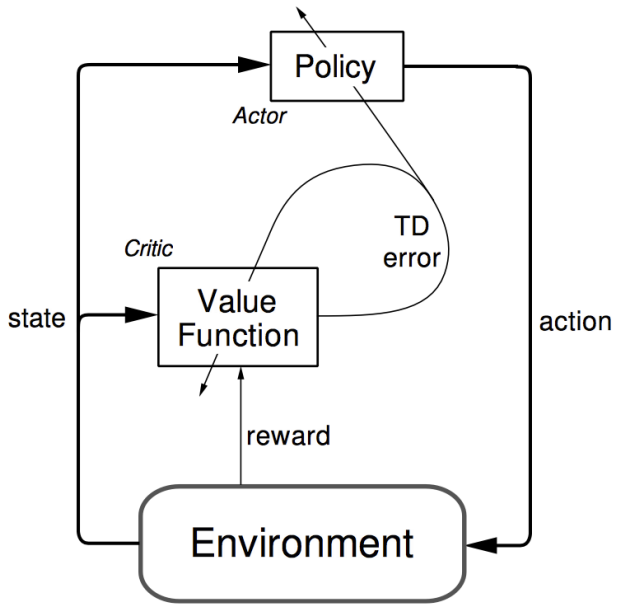
Reward is on all transitions -1 except those into the the region marked "The Cliff."



Q-learning learns quickly values for the optimal policy, that which travels right along the edge of the cliff. Unfortunately, this results in its occasionally falling off the cliff because of the ε-greedy action selection.
Sarsa takes the action selection into account and learns the longer but safer path through the upper part of the grid.

If ε were gradually reduced, then both methods would asymptotically converge to the optimal policy.

Actor-Critic Methods



- Explicit representation of policy as well as value function
- Critic drives all learning
- On policy method
- Appealing as psychological and neural models

Actor-Critic Details

Typically, the critic is a state-value function. After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected. That evaluation is the TD error:

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

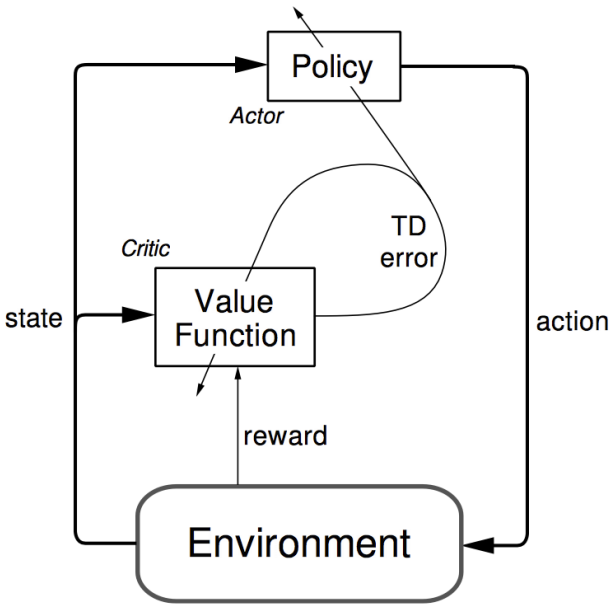
Let's assume actions are determined by preferences, p(*s*,*a*), as follows:

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

Then strengthening or weakening the preferences, p(*s*,*a*), depends on the TD error (β – step size parameter):

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

Actor-Critic Methods



Advantages:

- Minimal computation to select actions, since it does not have to search through the action space – particularly important for large action spaces.
- Can learn an explicit stochastic policy

R-Learning for Undiscounted Continuing Tasks

- off-policy control method
- no discounts
- no division of experience into distinct episodes with finite returns
- one seeks to obtain the maximum reward per time step

R-Learning

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

nonzero probability of reaching any state from any other under any policy

From any state, in the long run the average reward is the same, but there is a transient. From some states better-than-average rewards are received for a while, and from others worse-than-average rewards are received. It is this transient that defines the value of a state:

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

relative values because they are relative to the average reward under the current policy

Value of the state action pair:

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

R-Learning

- Other than its use of relative values, R-learning is a standard TD method using off-policy Generalized Policy Iteration (GPI), much like Q-learning.

It maintains:

- a behavior policy to generate experience, e.g. ε-greedy policy
- estimation policy, involved in GPI
- action value function
- estimated average reward

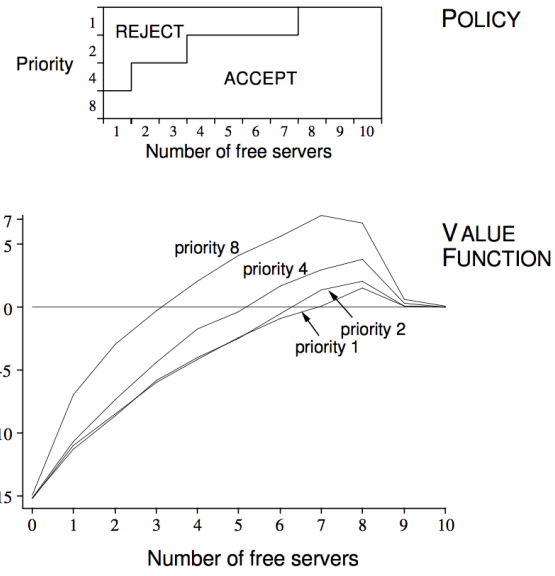
Initialize ρ and $Q(s,a)$, for all s, a, arbitrarily
Repeat forever:
  $s \leftarrow$ current state
  Choose action a in s using behavior policy (e.g., ε-greedy)
  Take action a, observe r, s'
  $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r - \rho + \max_{a'} Q(s',a') - Q(s,a)]$
  If $Q(s,a) = \max_a Q(s,a)$,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

Access-Control Queuing Task

- *n* servers
- Customers have four different priorities, which pay reward of 1, 2, 4, or 8, 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



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, Actor critic
  - Off-policy control: Q-learning and R-learning
- These methods bootstrap and sample, combining aspects of DP and MC methods

Unified View