Reinforcement Learning (INF11010)

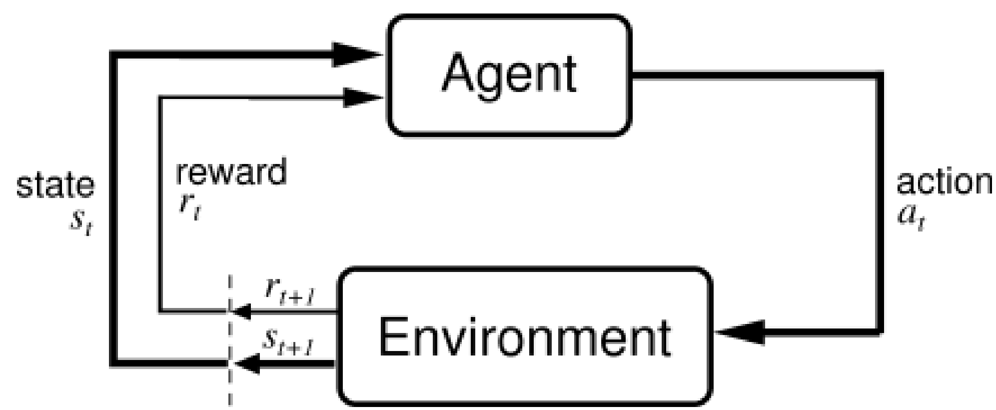
Lecture 8: Off-Policy Monte Carlo / TD Prediction

Pavlos Andreadis, February 13th 2018
with slides by Subramanian Ramamoorthy, 2017

Markov Decision Processes

- A finite Markov Decision Process (MDP) is a tuple (S, A, P, R, γ) where:

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability function
- R is a reward function
- γ is a discount factor



Methods Overview

- Dynamic Programming Methods:
  - *require a model*
  - *bootstrap*
- Monte Carlo Methods:
  - *do not* require a model
  - *do not* bootstrap
- Temporal-Difference Learning Methods:
  - *do not* require a model
  - *bootstrap*

Today's Content

- Off-Policy Monte Carlo
  - Incremental Implementation
- Temporal Difference Learning – Prediction
  - TD(0)

Off-policy Method

- Evaluate one policy while following another one
  - Behaviour policy takes you around the environment
  - Estimation policy is what you are after
- Of course, this requires: $\pi(s,a) > 0 \implies \pi'(s,a) > 0, \forall s, \forall a$
- Then, the off-policy procedure works as follows:
  - Compute the weighted average of returns from behaviour policy
  - Weighting factors are the probability of the moves being in estimation policy
  - i.e., weight each return by relative probability of being generated by π and π'

Learning a Policy while Following Another

On the ith first visit to state s, let:

$p'_i(s)$ = probability of getting subsequent sequence of states and actions from π' (BEHAVIOUR) T is the end-of-episode time

Using this to get data

$$p'_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$R'_i(s)$ = return observed from following the behaviour policy through this sequence of states and actions

Learning a Policy while Following Another

Let $p_i(s)$ = probability of getting the same sequence of states and actions from π (ESTIMATION)

$$p_i(s_t) \quad = \quad \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

Then after n_s returns experienced from state s (so episodes in which s occurs), weight each return by relative probability of occurring in π and π' and average:

$$V^\pi(s) \quad = \quad \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i'(s)} R_i'(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i'(s)}}$$

Comparing the two Probabilities

$$p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$$p'_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$$\frac{p_i(s_t)}{p'_i(s_t)} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

So the weighting factors don't depend on environment, only on the two policies. How can we use this?

Off-Policy MC Algorithm

How to use this formula to get Q-values?

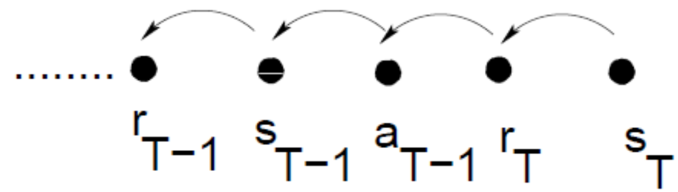
- Use *Behaviour Policy* π' to generate moves
  - must be soft so that all (s, a) continue to be explored
- Evaluate and improve *Estimation Policy* π
  - converges to optimal policy

So...

1. BP π' generates episode
2. EP π is deterministic and gives the greedy actions w.r.t. the current estimate of Q^{π} (it is arbitrary for the first episode)

Off-Policy MC Algorithm, cont.

3. Start at end of episode, work backwards



till BP and EP give divergent actions, e.g. back to time t

4. For this chain of states and actions compute

$$\frac{p_i(s_t)}{p_i'(s_t)} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

π is deterministic so $\pi(s_k, a_k)$ etc. $= 1$ and we know π'

Off-Policy MC Algorithm, cont.

So

$$\frac{p_i(s_t)}{p'_i(s_t)} = \prod_{k=t}^{T_i(s)-1} \frac{1}{\pi'(s_k, a_k)}$$

5.

$$Q(s,a) = \frac{\sum \frac{p_i}{p'_i} R'}{\sum \frac{p_i}{p'_i}}$$

Sum is over no. times this (s, a) has been visited, say N

R' = return for the chain of states/actions (see 3) following (s, a) (it's different for each of the N visits, as is p/p')

Off-Policy MC Algorithm, cont.

6. Do for each (s, a) in chain (see 3)

7. Improve π (estimation policy) to be greedy w.r.t. Q:

   $\pi(s) = \arg\max_a Q(s, a)$

   (Still deterministic, so still 1 for transitions within it.)

8. Back to 1. Repeat until estimation policy and Q values converge.

Takes a long time because we can only use the information from the end of the episode in each iteration.

The Off-Policy MC Control Algorithm

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$N(s,a) \leftarrow 0$; Numerator and

$D(s,a) \leftarrow 0$; Denominator of $Q(s,a)$

$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:

(a) Select a policy π' and use it to generate an episode:

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$$

(b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$

(c) For each pair s, a appearing in the episode after τ:

$t \leftarrow$ the time of first occurrence (after τ) of s, a

$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)}$

$N(s,a) \leftarrow N(s,a) + wR_t$

$D(s,a) \leftarrow D(s,a) + w$

$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$

(d) For each $s \in \mathcal{S}$:

$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Incremental Implementation

- Better to implement MC incrementally (think memory…)

- To compute the weighted average of each return:

$$V_n = \frac{\sum_{k=1}^{n} w_k R_k}{\sum_{k=1}^{n} w_k}$$

non-incremental

$$V_{n+1} = V_n + \frac{w_{n+1}}{W_{n+1}}\left[R_{n+1} - V_n\right]$$

$$W_{n+1} = W_n + w_{n+1}$$

$$V_0 = W_0 = 0$$

incremental equivalent

We may also wish to assign relative weights to different episodes…

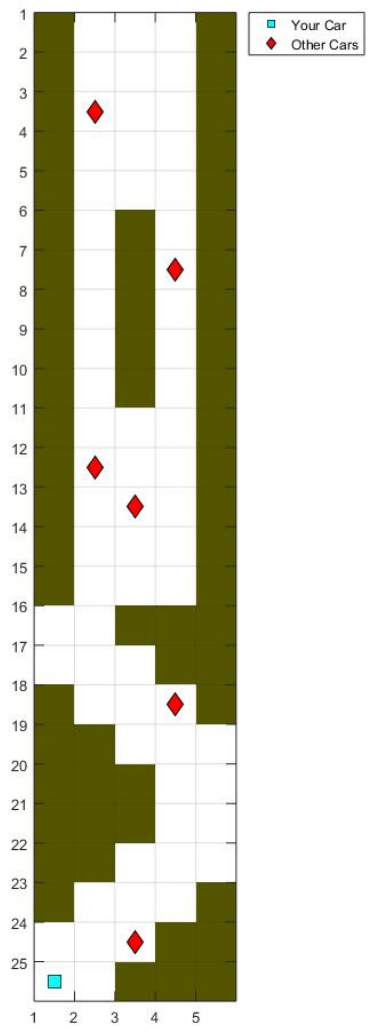
Road Fighter example

- Actions:

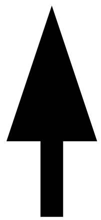
  UP_LEFT

  UP

  UP_RIGHT



- Policy for Evaluation:
  - Always UP

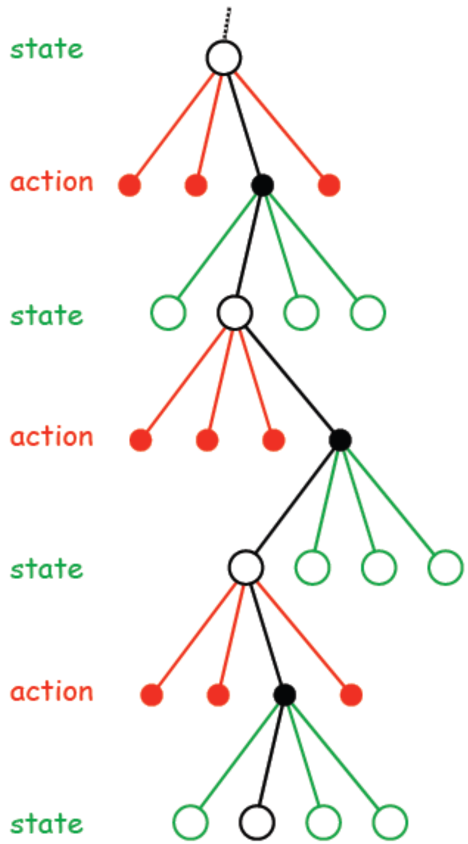


- Behavioural Policy:
  - $\pi(s,a) = 1/3$

Monte Carlo Summary

- Learn value functions and optimal policies for *sample episodes*
  - directly from interaction with environment
  - from simulator or *sample model*
  - can focus on subset of states
  - less harmed by violations of the Markov property
- Through the lens of Generalised Policy Iteration
  - different policy evaluation procedure
- For sufficient exploration
  - exploring starts (maybe in simulations; rarely in real life)
  - on-policy (best policy that still explores)
  - off-policy (decouples exploration from evaluated policy)

Learning in MDPs



- You are learning from a long stream of experience:

  $$s_0 a_0 r_0 s_1 a_1 r_1 \ldots s_k a_k r_k \ldots$$

  … up to some terminal state
- Direct methods:
  Approximate value function (V/Q) straight away - without computing $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$

Should you wait until episodes end or can you learn on-line?

Recap: Incremental Monte Carlo Algorithm

- Incremental sample-average procedure:

$$V(s) \leftarrow V(s) + \frac{1}{n(s)}[R - V(s)]$$

- Where $n(s)$ is number of first visits to state s
  - Note that we make one update, for each state, per episode
- One could pose this as a generic constant step-size algorithm:

$$V(s) \leftarrow V(s) + \alpha[R - V(s)]$$

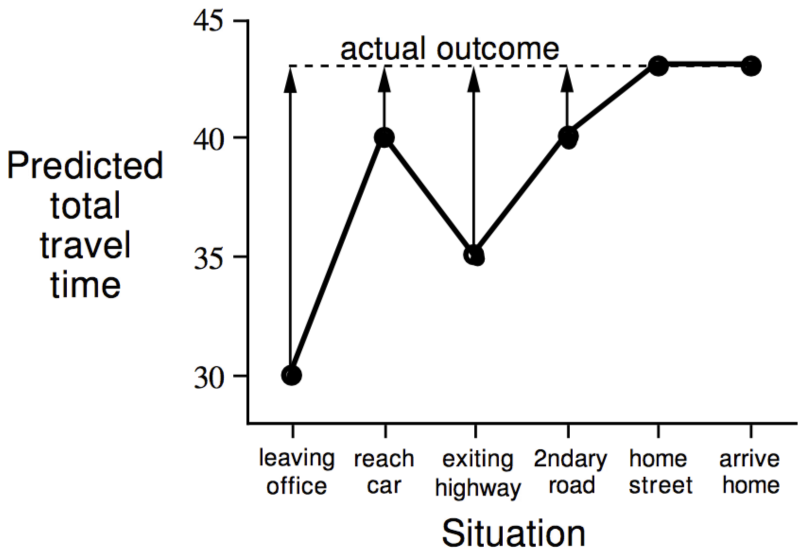
  - Useful in tracking non-stationary problems (task + environment)

Example: Driving Home

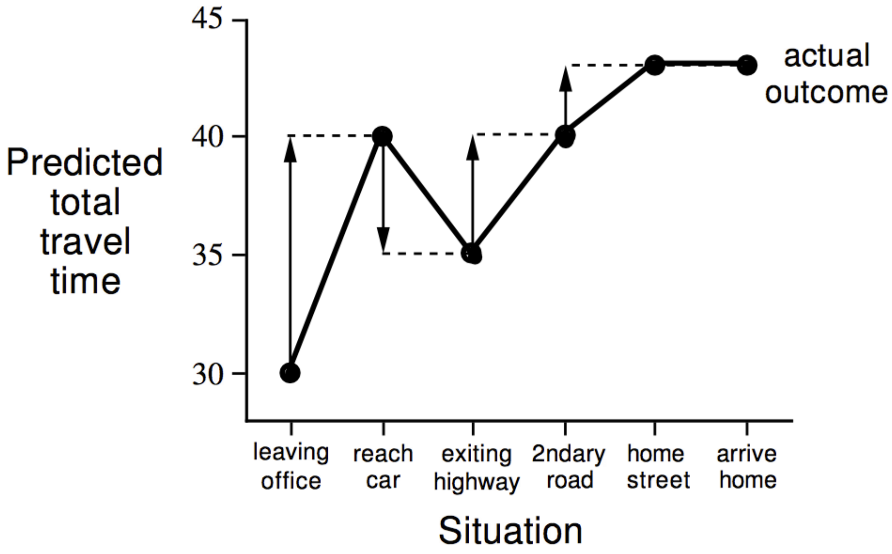
State	Elapsed Time (minutes)		Predicted Time to Go	Predicted Total Time
leaving office	0		30	30
reach car, raining	5	(5)	35	40
exit highway	20	(15)	15	35
behind truck	30	(10)	10	40
home street	40	(10)	3	43
arrive home	43	(3)	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)

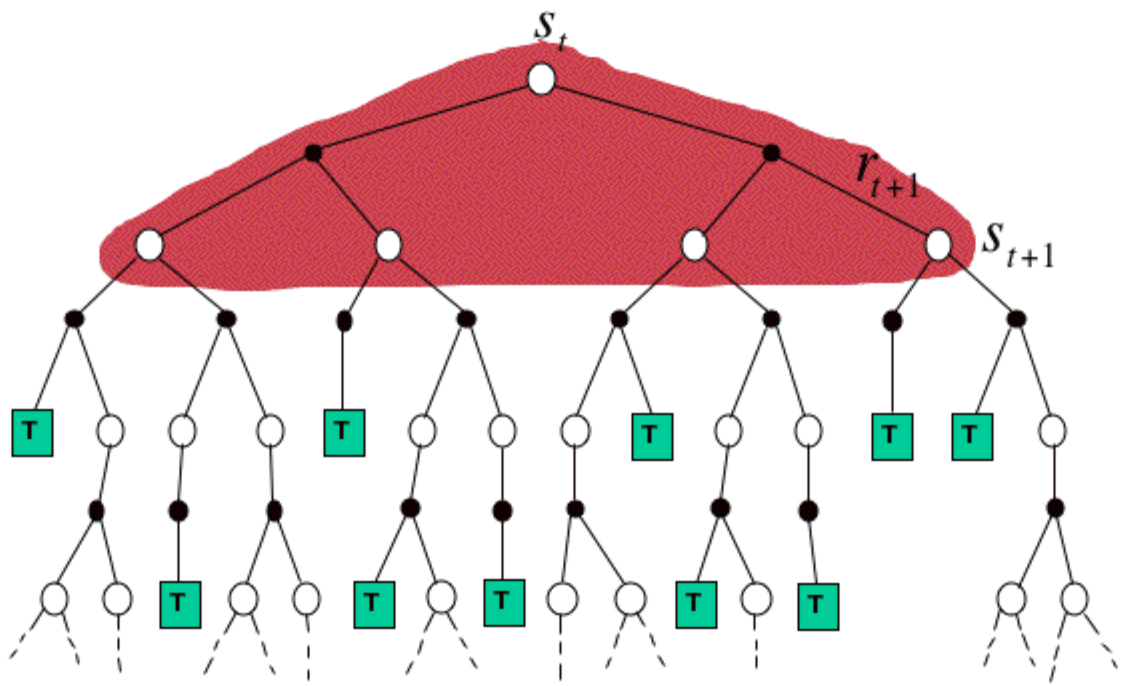


Changes recommended by TD methods (α=1)



What does DP Do?

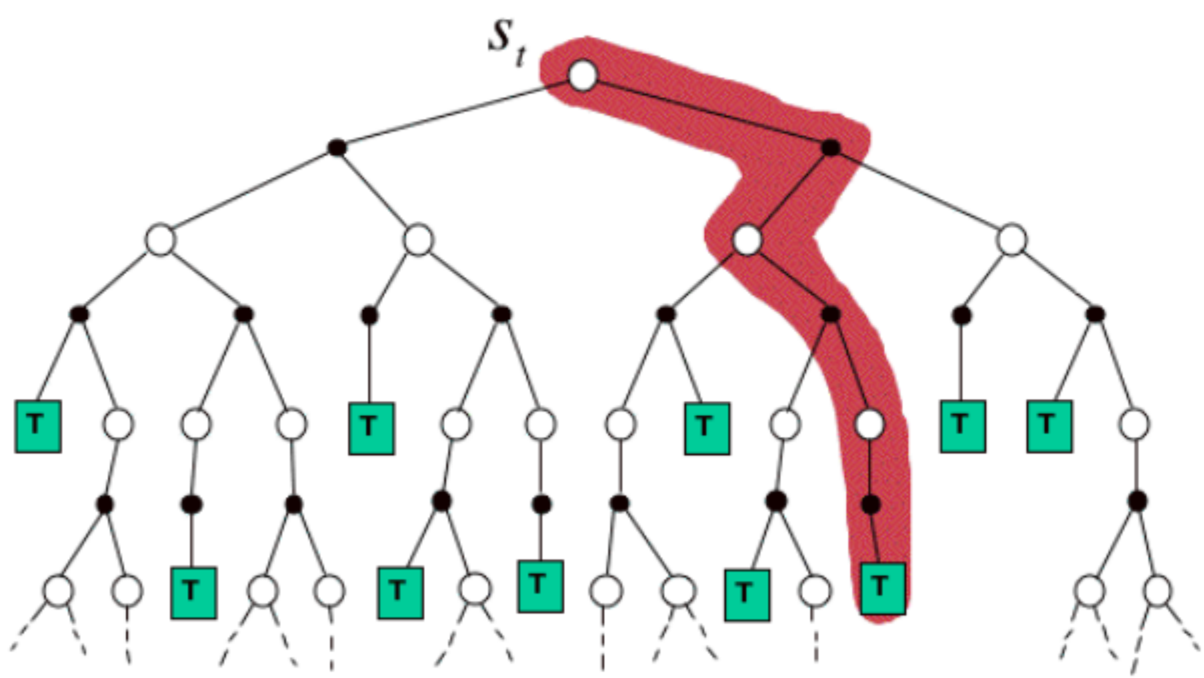
$$V(s_t) \leftarrow E_\pi\left\{r_{t+1} + \gamma\, V(s_t)\right\}$$



What does Simple MC Do?

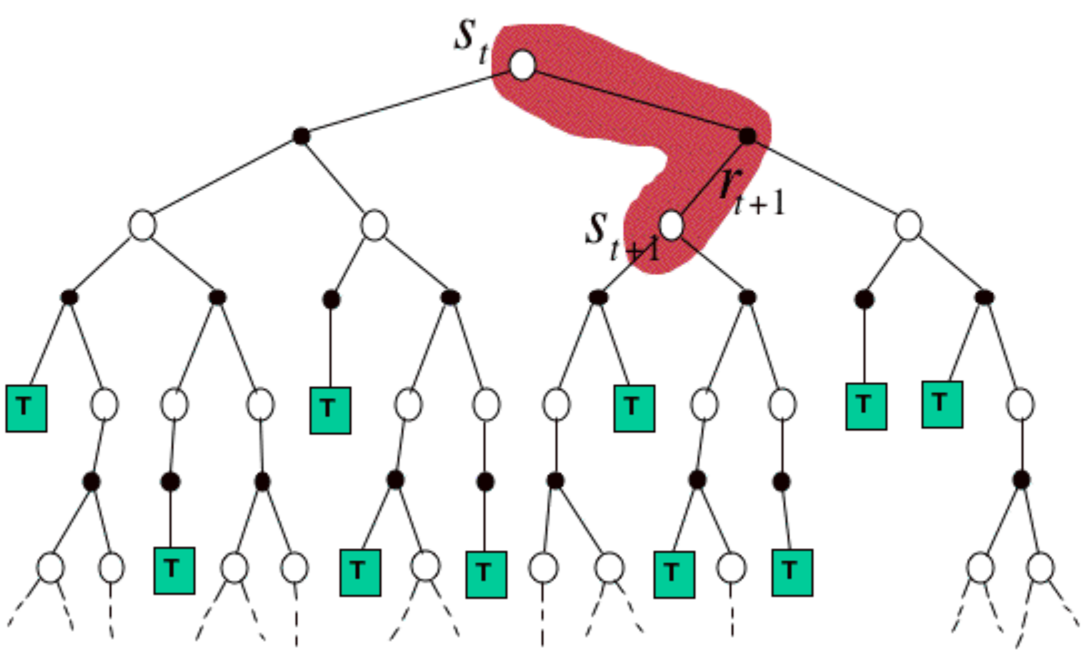
$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



Idea behind Temporal Difference Procedure

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



Temporal Difference Prediction

Policy Evaluation is often referred to as the Prediction Problem: we are trying to predict how much return we'll get from being in state s and following policy π by learning the state-value function V^π. Compare:

Monte-Carlo update:

$$V(s_t) \to V(s_t) + \alpha[\underbrace{R_t}_{\text{Target: actual return from } s_t \text{ to end of episode}} - V(s_t)]$$

Simplest temporal difference update TD(0):

$$V(s_t) \to V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{Target: estimate of the return}} - V(s_t)]$$

Both have the same form

Temporal Difference Learning

- *Does not* require a model (i.e., transition and reward prob.) – learn directly from experience
- Update estimate of *V(s)* soon after visiting the state *s*

$$\Delta V_t(s_t) = \alpha[r_{t+1} + \gamma V_{t+1}(s_{t+1}) - V_t(s_t)]$$

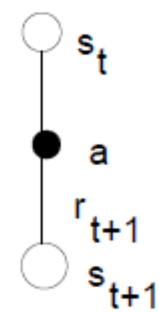
$$V_{t+1}(s_t) - V_t(s_t)$$

Actual 1-step reward

Discounted estimate of future reward

Initial estimate of future reward

This is better than .... this



Backup diagram

TD(0) Update

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

cf Dynamic Programming update:

$$\begin{aligned} V^\pi(s) &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} \\ &= \sum_a \pi(s,a,) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

$r_{t+1} + \gamma V(s_{t+1})$ is a better estimate of the value function than $V(s_t)$ because it replaces one step of estimated reward – that from t to $t+1$ – with the **actual** reward r_{t+1} obtained in that step.

TD(0) Algorithm for Learning V^{π}

- Initialise $V(s)$ arbitrarily; π is the policy to be evaluated; choose learning rate α and discount factor γ
- Repeat for each episode

  Pick a start state s

  Repeat for each step in episode

  Get action a given by policy π for state 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

From S+B Fig. 6.1

Why TD Learning?

- Don't need a model of the environment
- On-line and incremental – updates each step – so can be fast

  don't need to wait till the end of the episode so need less memory, computation

  subsequent updates take immediate advantage of updated values

  cf. Monte Carlo – waits till end of episode, episodes may be long or tasks continuing, some MC must ignore episodes with exploratory steps
- Updates are based on actual experience (r_{t+1})
- Converges to $V^\pi(s)$ – but must decrease step size α as learning continues

Why?

Bootstrapping, Sampling

TD **bootstraps**: it updates its estimates of V based on other estimates of V

DP also bootstraps

MC does not bootstrap: estimates of complete returns are made at the end of the episode

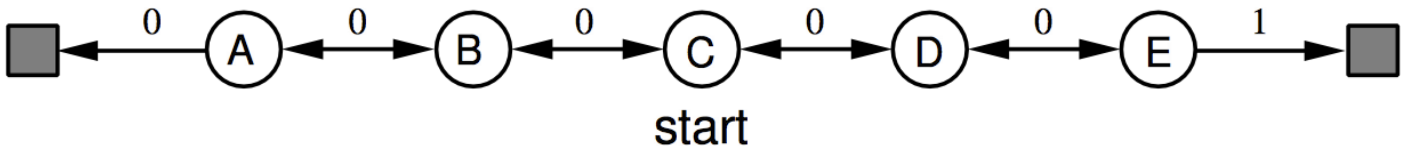
TD **samples**: its updates are based on one path through the state space

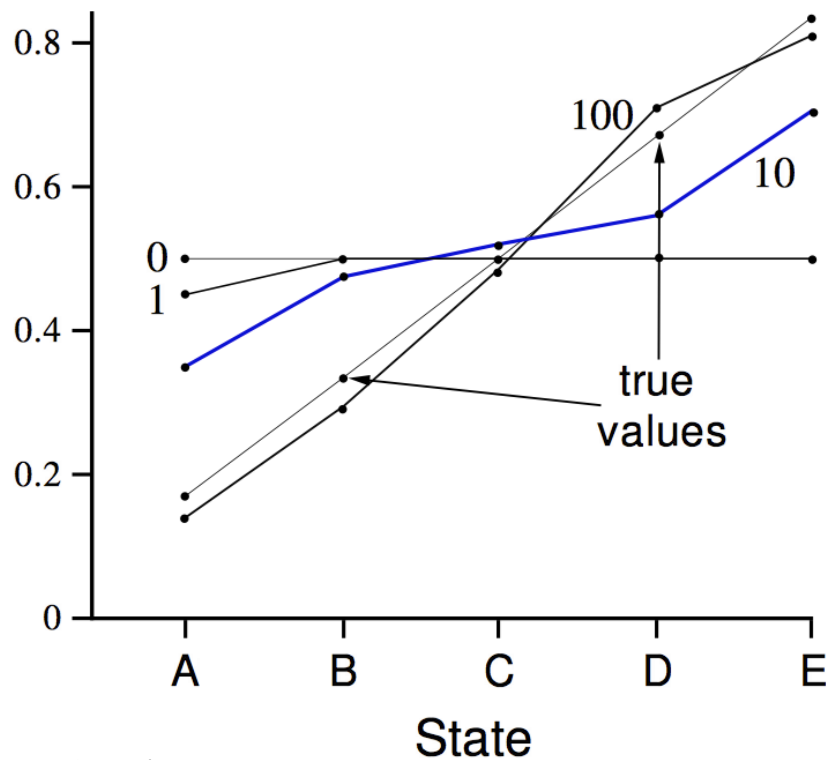
MC also samples

DP does not sample: its updates are based on all actions and all states that can be reached from the updating state

Examples: see e.g. random walk example S+B sect. 6.2

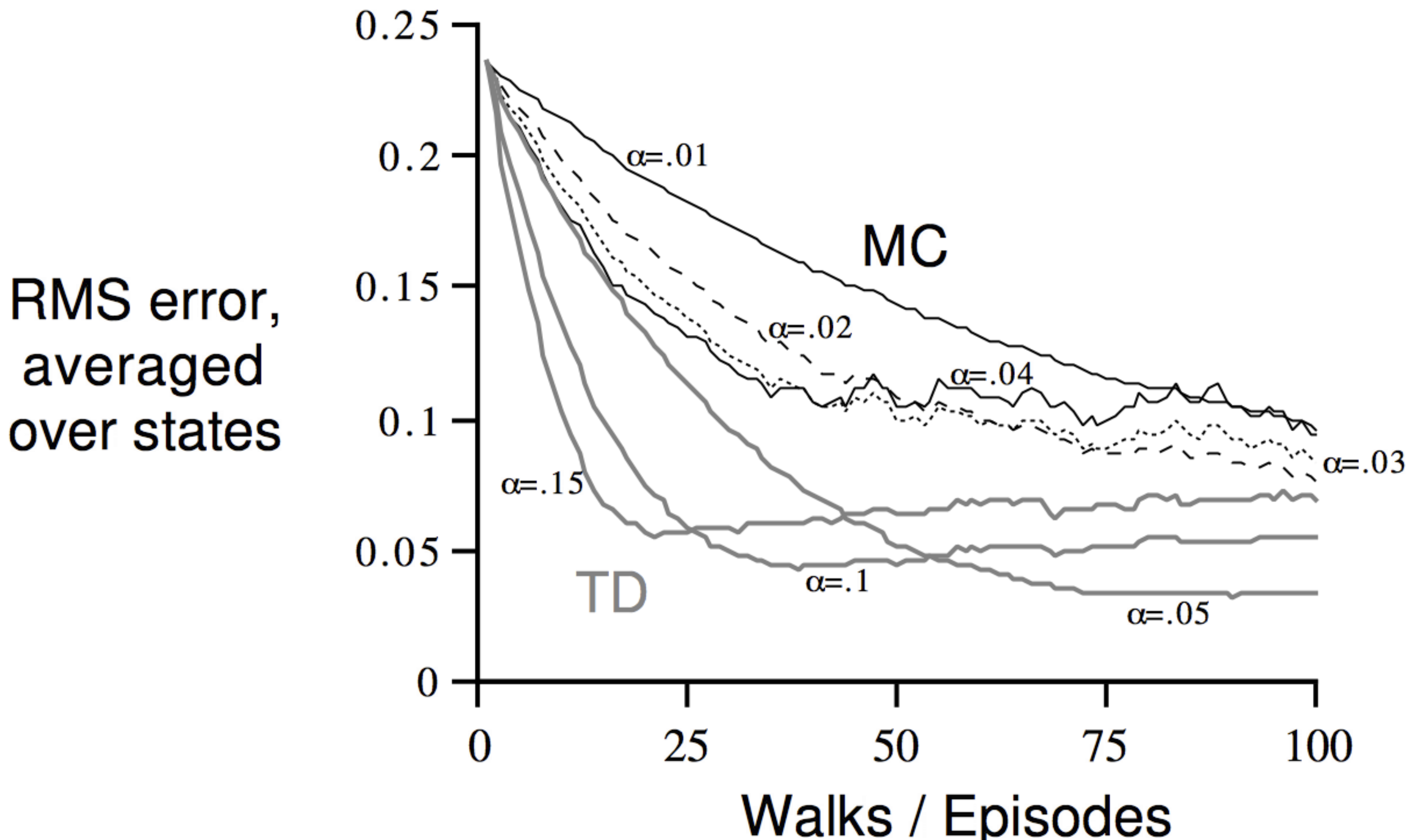
Random Walk Example





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

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Understanding TD vs. MC

S+B Example 6.4:

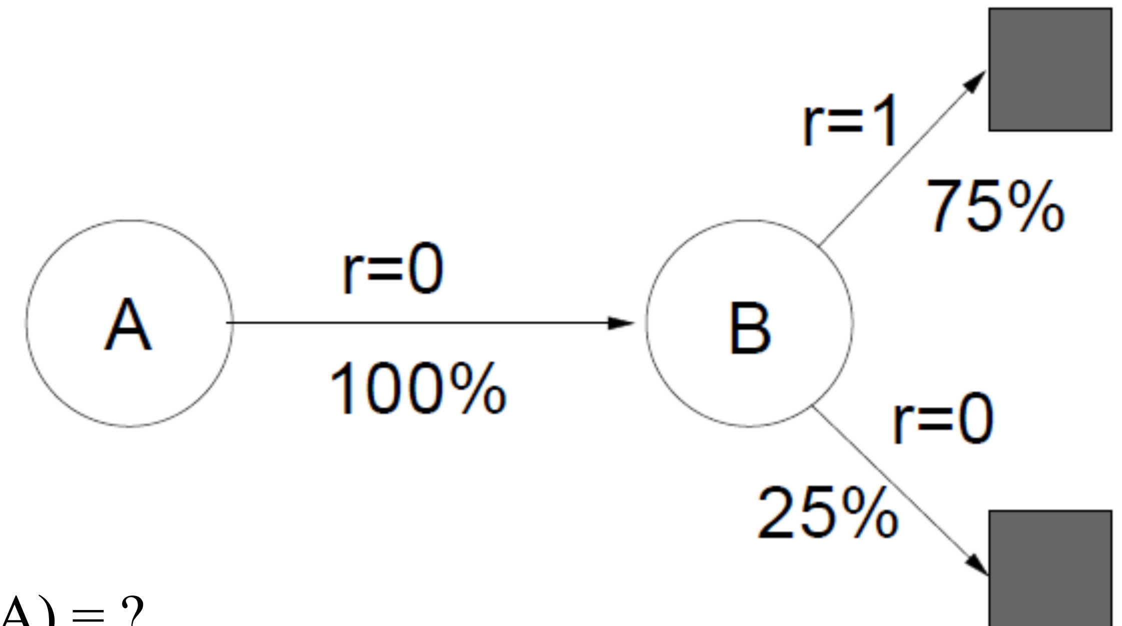
- You observe 8 episodes of a process:

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

- Interpretation:
  - First episode starts in state A, transitions to B getting a reward of 0, and terminates with a reward of 0
  - Second episode starts in state B, terminates with a reward of 1, etc.

<u>Question</u>: What are good estimates for $V(A)$ and $V(B)$?

S+B Example 6.4: Underlying Markov Process



V(A) = ?

TD and MC Estimated

- Batch Monte Carlo (update after all episodes done) gets V(A) = 0.
  - This best matches the training data
  - It minimises the mean-square error on the training set
- Consider sequentiality: A to B to terminating state; V(A) = 0.75.
  - This is what TD(0) gets
  - Expect that this will produce better estimate of future data even though MC gives the best estimate on the present data
  - Is correct for the maximum-likelihood estimate of the model of the Markov process that generates the data, i.e. the best-fit Markov model based on the observed transitions
  - Assume this model is correct; estimate the value function – "certainty-equivalence estimate"

TD(0) tends to converge faster because it moves towards a *better* estimate.

Reading +

- Chapter 5 (5.5 to end) and Chapter 6 (6.1 to 6.3) of Sutton and Barto (1st Edition) http://incompleteideas.net/book/ebook/the-book.html

Optional (weighting the update for MC):

- *Section 5.5 of Sutton and Barto (2nd Edition)* http://incompleteideas.net/book/bookdraft2018jan1.pdf