Temporal Difference Learning:
Bootstrapping Value Functions

Prof. Matthias Grossglauser

Information and Network Dynamics (INDY) lab
School of Computer and Communication Sciences (I&C)
EPFL

Recap

- Markov Decision Process (MDP): states that capture everything we need to know about the past, actions that can only depend on states
- Bellman equations: expresses recursive consistency of value function

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

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] = \max_a q_*(s,a)$$

$$q_\pi(s,a) = \sum_{s',\boldsymbol{r}} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s')\, q_\pi(s',a') \right] = \sum_{s',\boldsymbol{r}} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right] = \sum_{\boldsymbol{s'},r} p(s',r|s,a)[r + \gamma v_*(s')]$$

Recap: dynamic programming

- Computing the value function of policy π:
  - $v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$

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

  - Stop when change in v_k is negligible. $v_k \to v_\pi$
- Policy Improvement:
  - $v_{\pi'}(s) \coloneqq \max_{a \in \mathcal{A}} q_\pi(s, a) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$
  - $\pi'(s) \coloneqq \arg\max_{a \in \mathcal{A}} q_\pi(s, a) = \arg\max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$
- Policy Iteration: $\pi_0 \xrightarrow{\text{eval}} v_{\pi_0} \xrightarrow{\text{impr}} \pi_1 \xrightarrow{\text{eval}} v_{\pi_1} \xrightarrow{\text{impr}} \pi_2 \xrightarrow{\text{eval}} v_{\pi_2} \xrightarrow{\text{impr}} \dots \xrightarrow{\text{impr}} \pi_* \xrightarrow{\text{eval}} v_* \xrightarrow{\text{impr}} \pi_*$

Extending DP to unknown environments

- Dynamic Programming assumes we know the MDP:
  - $p(s', r|s, a)$ known for all s, a, r, s'
- In real-world scenarios, the MDP is unknown
  - two-player games: we do not know the opponent's strategy
  - robot in the physical world: we cannot exactly model the forces and constraints
- We can learn from experiences
  - Via sampling and estimation
  - Monte Carlo methods did just that
    - Simple averaging of episodic rewards to estimate value functions
- Can we combine the strategies of dynamic programming and Monte Carlo to estimate better?
- That is exactly what we will do today: Temporal Difference (TD) Learning

The prediction problem

Goal: estimate $v_\pi(s)$ for all $s \in S$

- We want to adopt an approach like policy iteration…

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$$

- The expectation is over the joint distribution:

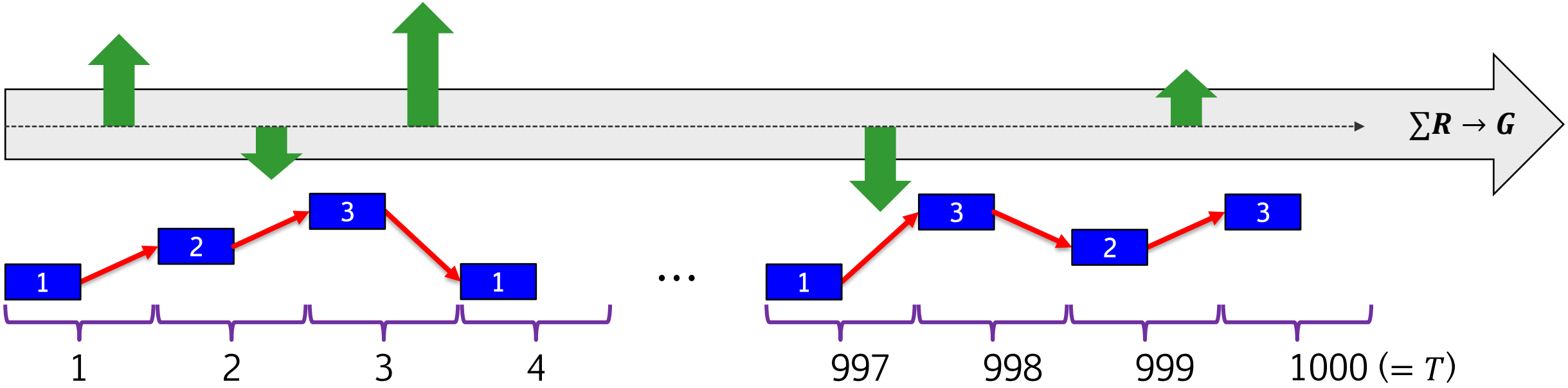
$$\pi(A_t \mid S_t = s) p(S_{t+1}, R_{t+1} | S_t = s, A_t)$$

  - **Note:** A_t is implicit in the above

- We do not have the probabilities ⇒ cannot compute expectation exactly
- We can get episodes (generated from π): $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- How can we approximate the expectation with this data?

An issue with MC: long episodes → infrequent updates

- Trajectory from one long episode → only one new estimate of G per visited state → one new sample for estimator of $v(s)$ (`return`$[s]$)

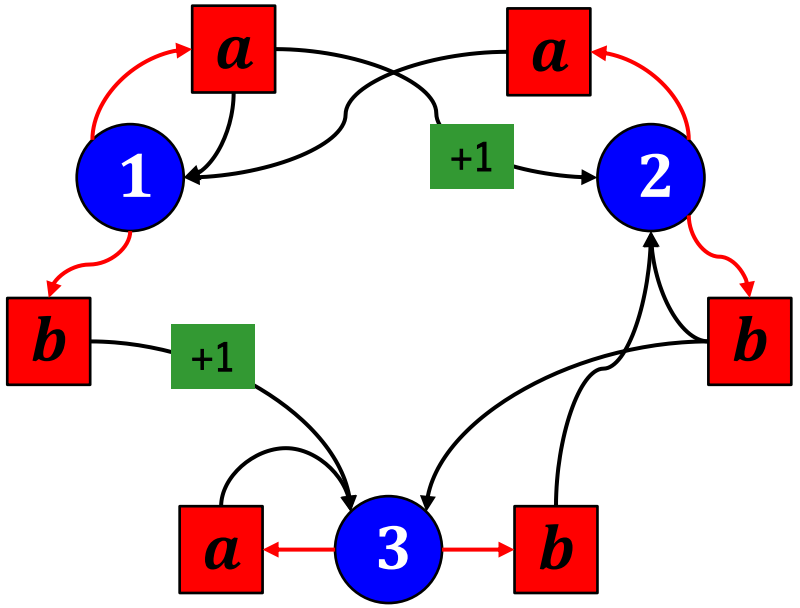


Key idea: bootstrapping

- Example: simple MDP to the right
- Suppose someone gave us estimates for $v(s)$ as follows:

$$\hat{v}(1) = 10$$
$$\hat{v}(2) = 10$$
$$\hat{v}(3) = 1000$$

- Would you trust these estimates?
  No, because from the Bellman equations, we know that value $v(3)$ should be slightly lower than value $v(2)$
- Bootstrapping: propagate information between states to improve all the estimators (and improve them faster than with MC)



Estimating from data

Goal: estimate $v_\pi(s)$ for all $s \in S$

- We want to approximate the update equation

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$$

- With episodic data generated from π: $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- Given $S_t = s$, (A_t, R_{t+1}, S_{t+1}) is a sample from the desired distribution:

$$\pi(A_t \mid S_t = s) p(S_{t+1}, R_{t+1} | S_t = s, A_t)$$

- First idea: across different episodes, every time $S_t = s$,
  - Set $v_{k+1}(s) = \text{average}(R_{t+1} + \gamma v_k(S_{t+1}))$
  - Iterate until convergence
- There is a better approach: a bit of aging of old information

An iterative averaging method

Let us revisit Monte-Carlo, where we estimate $v(s)$ by averaging episodic returns. Denote:

- v_n: the estimate of $v(s)$ by averaging over n episodes
- G_i: the episodic return of the ith episode

$$v_n = \frac{1}{n}\left(\sum_{i=1}^{n} G_i\right) = \frac{1}{n}\left(G_n + \sum_{i=1}^{n-1} G_i\right) = \frac{1}{n}\left(G_n + (n-1)v_{n-1}\right)$$
$$= v_{n-1} + \frac{1}{n}\left(G_n - v_{n-1}\right)$$

In-place, for all states: $v(s) \coloneqq v(s) + \frac{1}{n}\left(G_t - v(s)\right)$

This is an update of the form:

NewEstimate = OldEstimate + StepSize * [Target – OldEstimate]

Stochastic approximation updates

- More generally, $v_n = v_{n-1} + \alpha_n(G_n - v_{n-1})$
- For $\alpha_n = 1/n$ → v_n = avg(G_n) , as just seen
- For $\alpha_n = \alpha \in (0,1)$ (const.): $v_n = \alpha G_n + (1-\alpha)v_{n-1}$

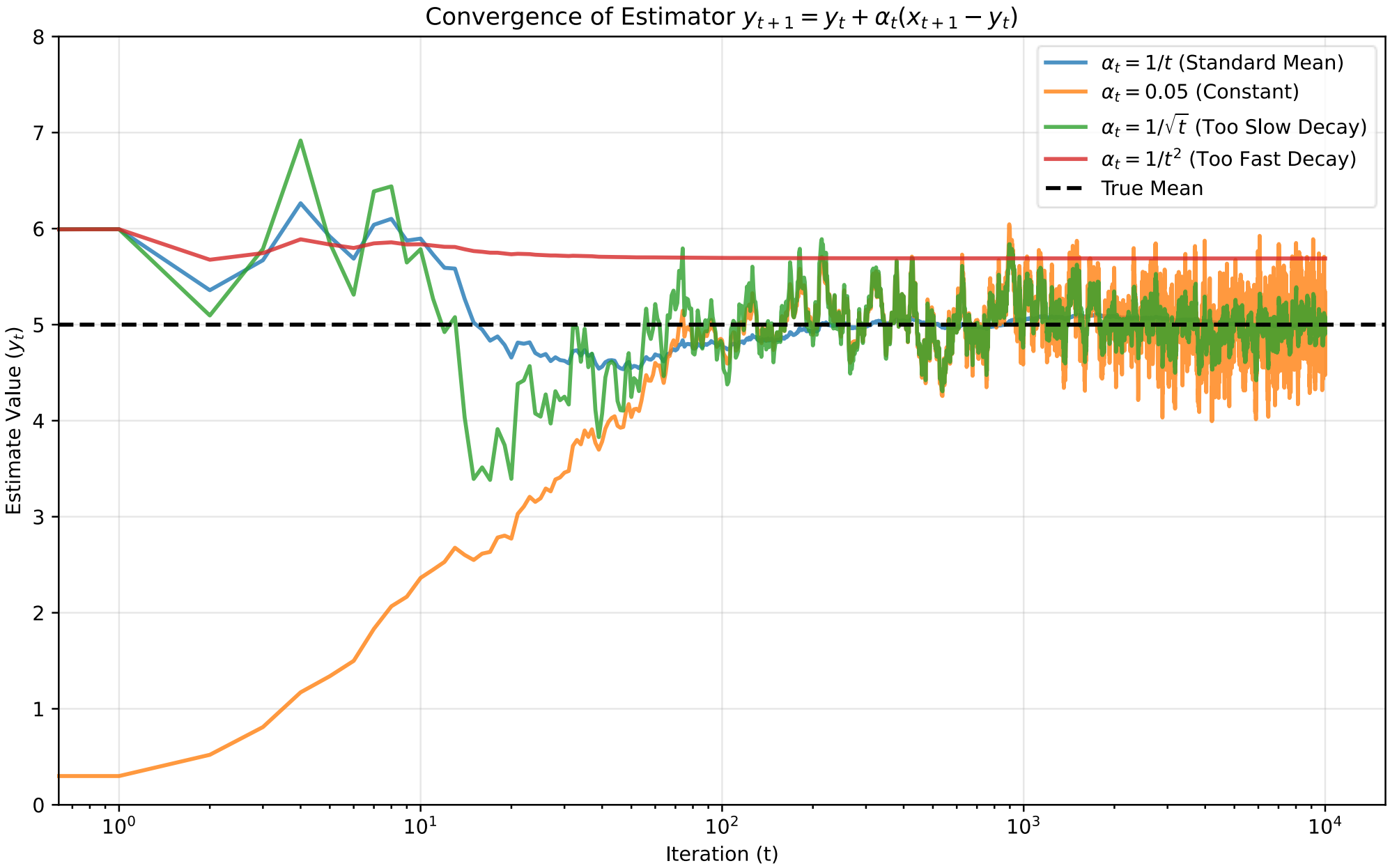
$$= \alpha G_n + \alpha(1-\alpha)G_{n-1} + (1-\alpha)^2 v_{n-2}$$
$$= \alpha G_n + \alpha(1-\alpha)G_{n-1} + \alpha(1-\alpha)^2 G_{n-2} + \alpha(1-\alpha)^3 v_{n-3}$$
$$= \alpha \sum_{k=0}^{\infty} (1-\alpha)^k G_{n-k}$$

  - Exponentially Weighted Moving Average (EWMA)
  - Sum of weights: $\alpha \sum_{k=0}^{\infty}(1-\alpha)^k = \frac{\alpha}{1-(1-\alpha)} = 1$
- Note: suppose $v_n \sim v$ is an i.i.d. sequence of RVs
  - Law of large numbers: v_n= avg(G_n) $\longrightarrow \mathbb{E}[v]$
  - EWMA: v_n does not converge to a constant; decrease in weight too fast → it "wiggles"permanently

Robbins-Monro conditions for convergence

- If $\alpha_n \to 0$ satisfies the following two conditions:
  (1) $\sum_{n=1}^{\infty} \alpha_n = \infty$
  (2) $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$
  then $v_n \longrightarrow \mathbb{E}[v]$ almost surely
- This holds even more generally than for i.i.d. sequences – e.g. for (bounded) ergodic Markov chains
- If (1) does not hold: updates too small → initial conditions (early samples) never "forgotten"
  - E.g. $\alpha_n = e^{-n}$
- If (2) does not hold: updates too large → no convergence
  - E.g. EWMA just seen; $\alpha_n = 1/\log n$
- In practice, we often violate (2) deliberately, because it allows to slowly track non-stationarities in the system (in exchange of a bit of noise)

Convergence with different α_n



The fundamental TD-learning update

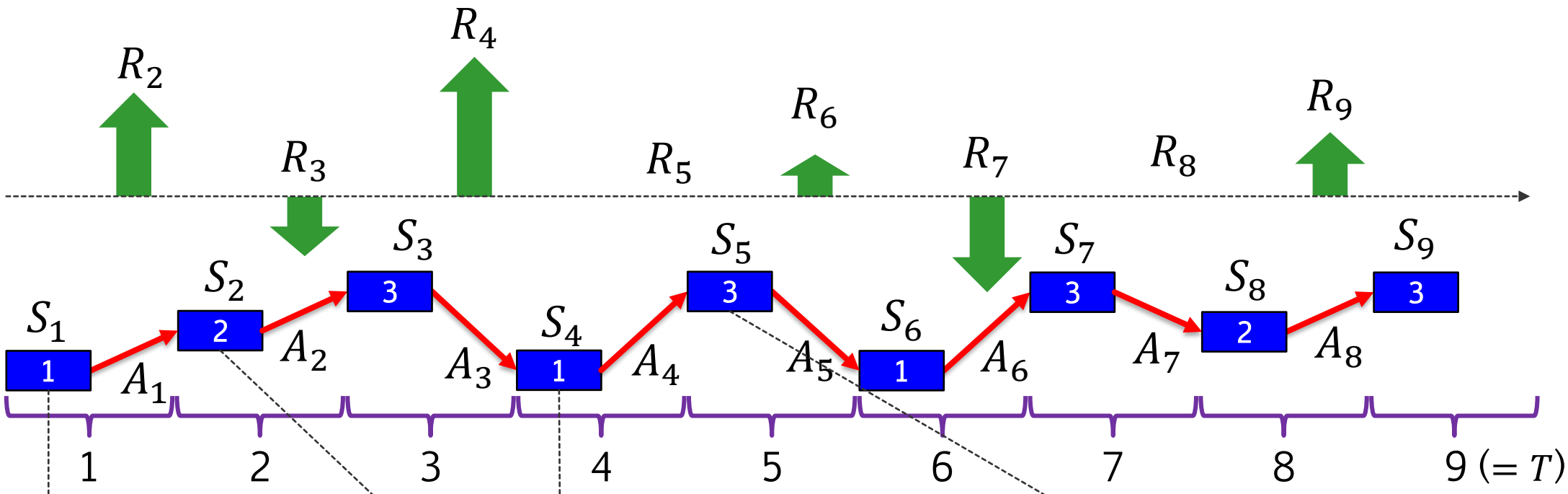
Given episode $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T,$

- The simplest TD-learning method makes the update

  $$v(S_t) \coloneqq v(S_t) + \alpha_n[R_{t+1} + \gamma v(S_{t+1}) - v(S_t)]$$

  immediately on transition to S_{t+1} and receiving R_{t+1}
- $R_{t+1} + \gamma v(S_{t+1})$ is the target for TD-learning
  - TD-target can be computed immediately, without unrolling full episode!
- In Monte Carlo, G_t is the target
  - Needs entire episode to be rolled out!
- For simplicity, keep $\alpha_n = \alpha \in (0, 1)$ as a constant step-size parameter
  - More on step-sizes later
- This algorithm is called TD(0)
  - General case: TD(λ), extrapolates between TD(0) and Monte Carlo (TD(1))

TD value prediction



$$v(1) \leftarrow v(1) + \alpha[R_2 + \gamma v(3) - v(1)]$$

$$v(1) \leftarrow v(1) + \alpha[R_2 + \gamma v(2) - v(1)]$$

Every state-to-state transition induces an estimator update
→ information "propagates" through the MDP network during the episode

TD(0) algorithm for policy evaluation

INPUT: the policy π to be evaluated

PARAMETER: step size $\alpha \in (0, 1]$

INITIALIZE: $v(s)$ arbitrarily for intermediate states, $v(\text{terminal}) = 0$

REPEAT: for each episode,

Pick an initial state S_0

FOR $t = 0, 1, 2, \ldots$:

sample A_t according to $\pi(S_t)$

Take action A_t, observe R_{t+1}, S_{t+1}

$v(S_t) \coloneqq v(S_t) + \alpha[R_{t+1} + \gamma v(S_{t+1}) - v(S_t)]$

UNTIL end of episode ($S_t = \otimes$)

TD = MC + DP

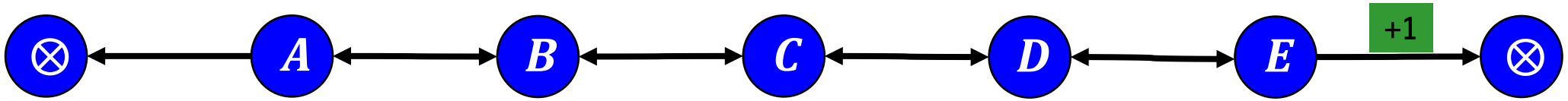
Temporal Difference learning is a neat combination of ideas from dynamic programming and Monte Carlo estimation.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] && (1)\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] && (2)
\end{aligned}
$$

- Monte Carlo methods use an estimate of (1) as target
  - Replacing actual expectation by empirical average from data
- Dynamic programming methods use an estimate of (2) as target
  - Replacing $v_\pi(S_{t+1})$ with an estimate $v_k(S_{t+1})$
- TD learning also uses an estimate of (2) as target
  - It uses averaging instead of true expectation (like MC)
  - And it uses an estimate $v_k(S_{t+1})$ instead of true value $v_\pi(S_{t+1})$ (like DP)

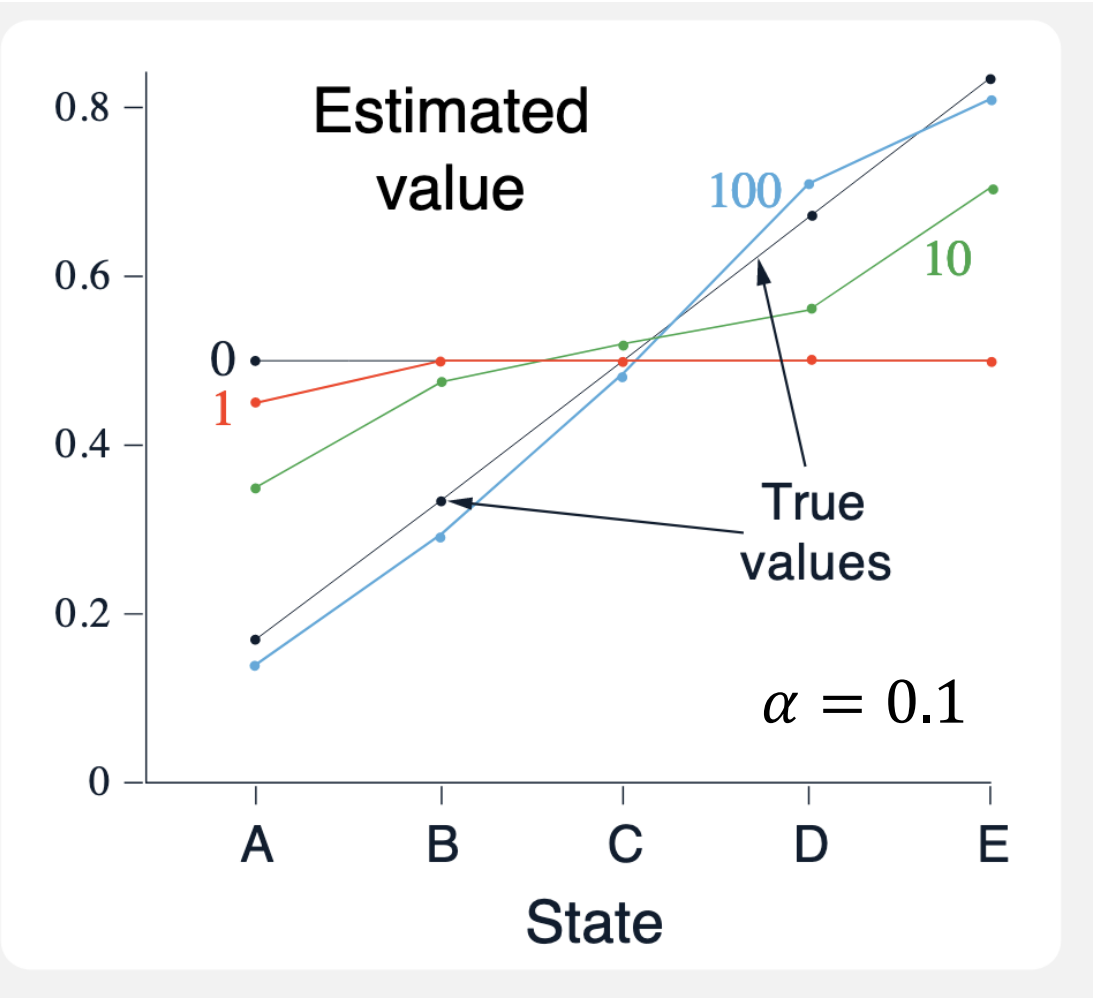
A random walk example

- Consider the following MDP with two terminal states and five intermediate states



- Policy π: move left or right at random, equal probability of both options
- Undiscounted rewards ⇒ true value of state s = probability of reaching right end, starting from state s
- Simple calculation (via DP) shows:
- $v_\pi(s) = \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$ for $s = A, B, C, D, E$ respectively
- How well can TD learn this value function?

Convergence of TD learning



The values learned after {1,10,100} episodes via TD(0).

- Initially, $V_0(s) = 0.5$ for all states
- With every run, through TD(0), value of some states updated
- With sufficient episodes, value function is learned to sufficient degree

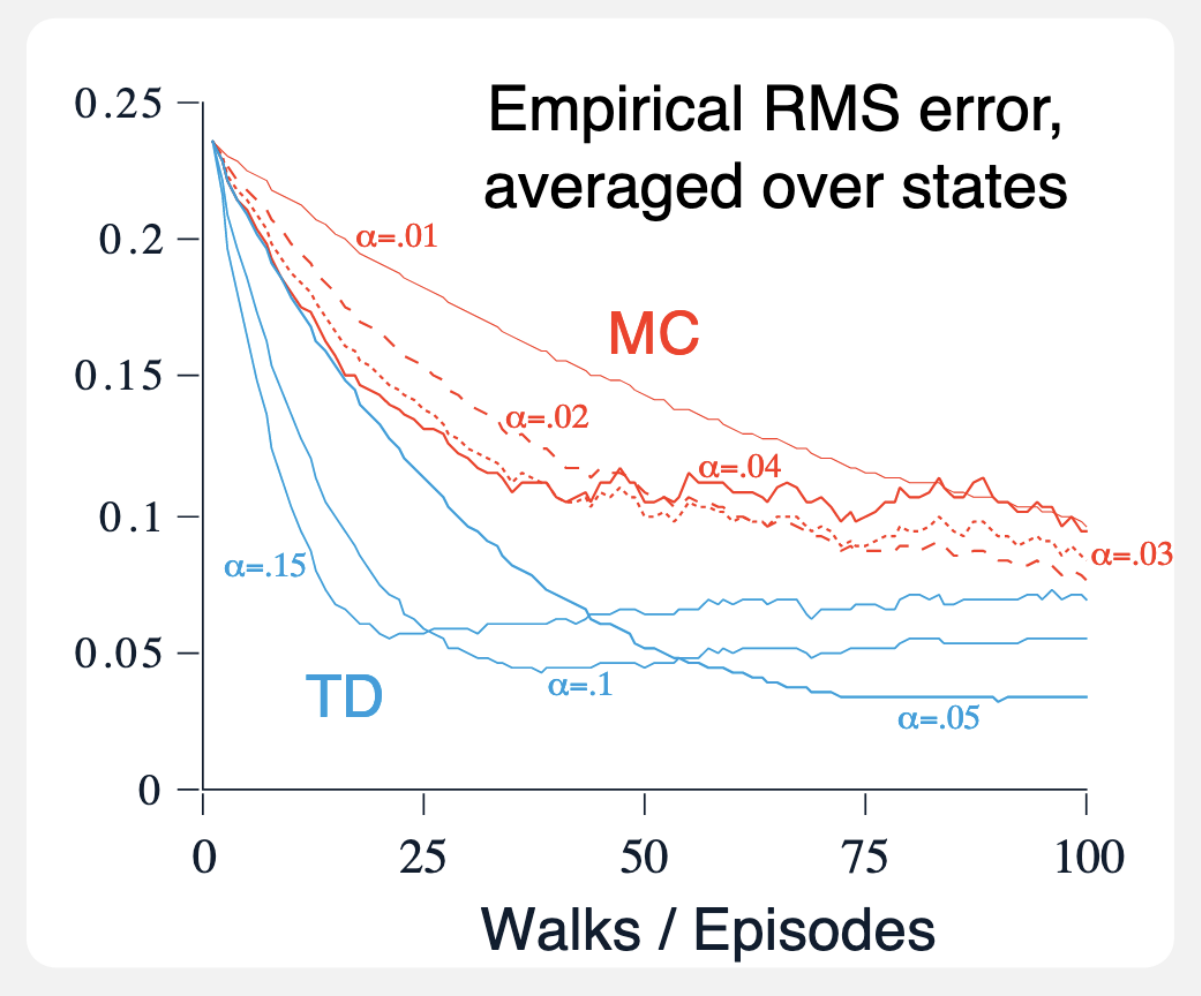
Root Mean Square Error (RMSE):

$$\sqrt{\frac{1}{|S|}\sum_{s\in S}\left(v_\pi(s) - V_k(s)\right)^2}$$

measures quality of approximation of black curve by red/green/blue curves

Note: With a constant step size $\alpha_n = \alpha$, the values fluctuate indefinitely in response to the outcomes of the most recent episodes

TD v/s MC



Across different values of α,

TD-learning performs better than Monte Carlo estimation

General trend as α decreases:

- Initial rate of decrease in error is slower
- Eventually converges to a lower error estimate
- Pattern seen in many machine learning problems (α: step-size or learning rate)

Does this mean TD learning always outperforms Monte Carlo methods?

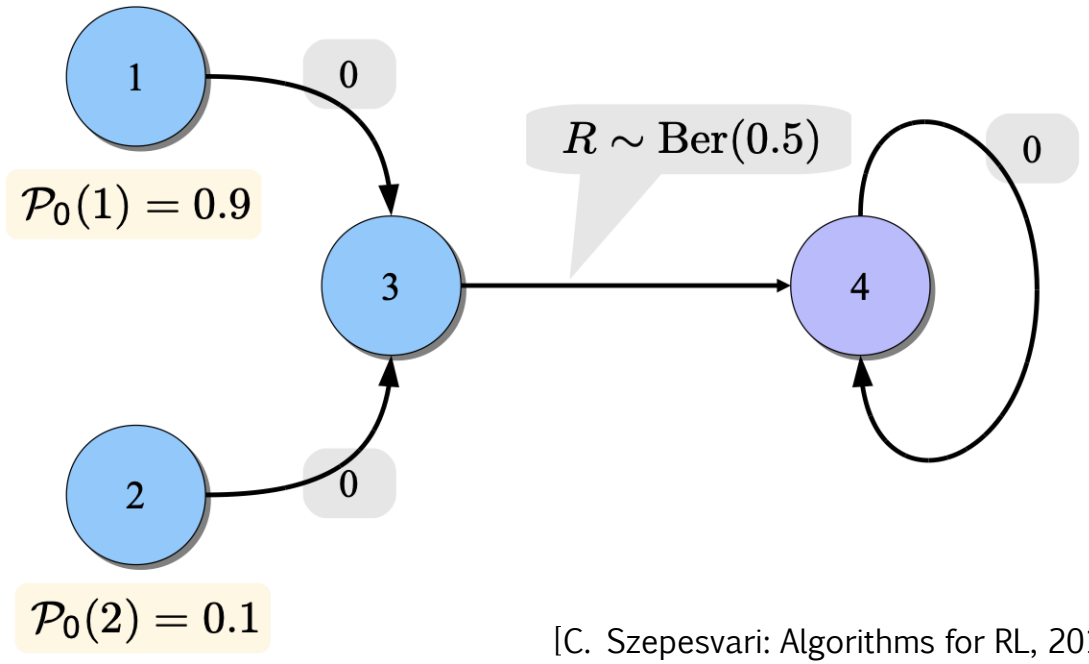
A second example

The following MDP has:

- Two possible starting states $(1, 2)$
- One intermediate state (3)
- And one terminal state (4)
- No discounting

State transitions are deterministic.
Each episode lasts two steps $(T = 2)$
Only sources of randomness are:

- Starting point S_0
- Reward R_2 in transition $3 \to 4$



[C. Szepesvari: Algorithms for RL, 2010]

Note: policy does not matter; only one possible state transition

Goal: estimate value function of states $(1, 2, 3)$ as quickly as possible

Here we can easily see: $v(1) = v(2) = v(3) = 0.5$

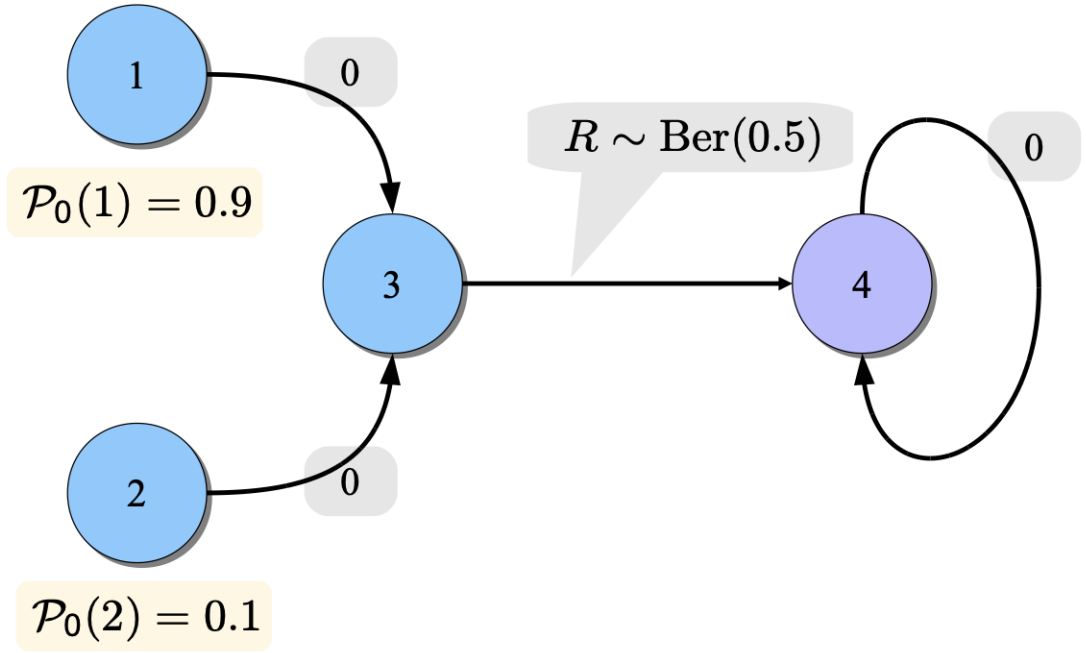
TD bootstraps, MC does not

Because of the structure of MDP,
after $10k$ episodes:

- $\approx 10k$ transitions $3 \to 4$
- $\approx 10k$ samples of the Bernoulli reward

$v(3)$ is the average of these $10k$ samples

- $\approx 9k$ starts from state 1
- $\approx k$ starts from state 2



In TD-learning, $v(2)$ is updated based on the estimate of $v(3)$

- $v(3)$ is average of $10k$ episodes → more accurate
- In estimating $v(2)$, we bootstrap from episodes that do not touch (2)

In Monte Carlo, $v(2)$ averages rewards of only those episodes that start from (2)

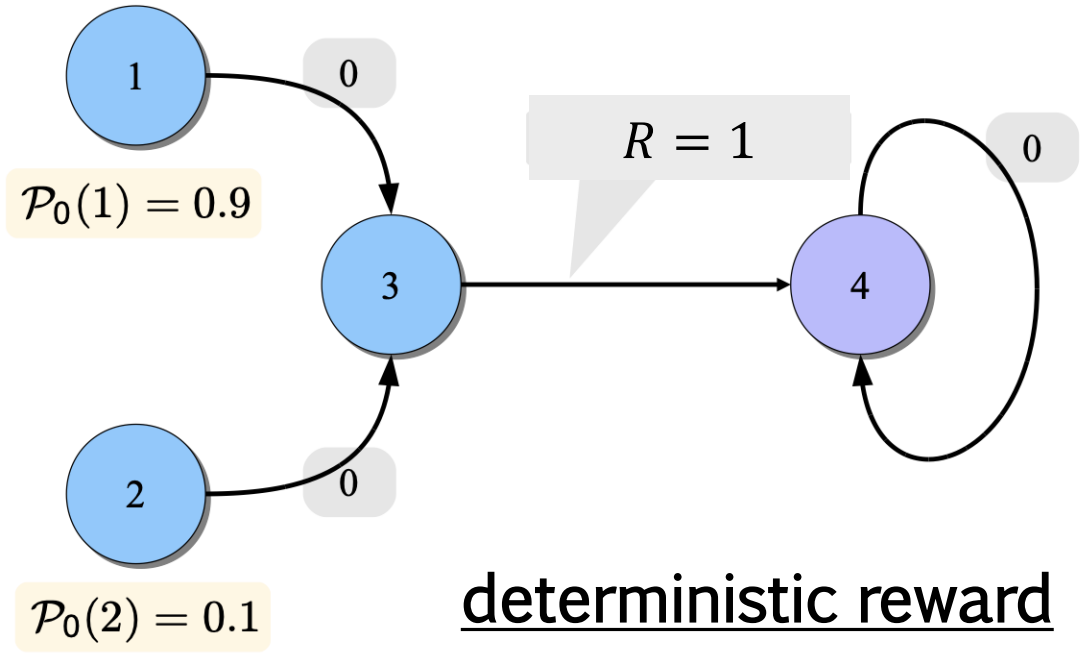
- No bootstrapping ⇒ less data ⇒ slower convergence

MC strikes back

Consider modification of earlier MDP where $3 \rightarrow 4$ transition gives a deterministic reward of 1

In Monte Carlo method, G_t is the target for $v(2)$

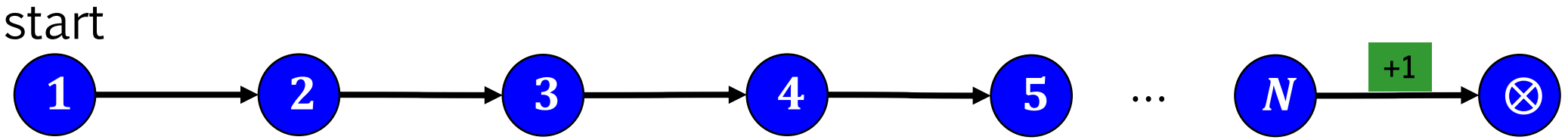
- $G_t = R_2$ is always 1
- $v(2)$=1 as soon as an episode starts at 2



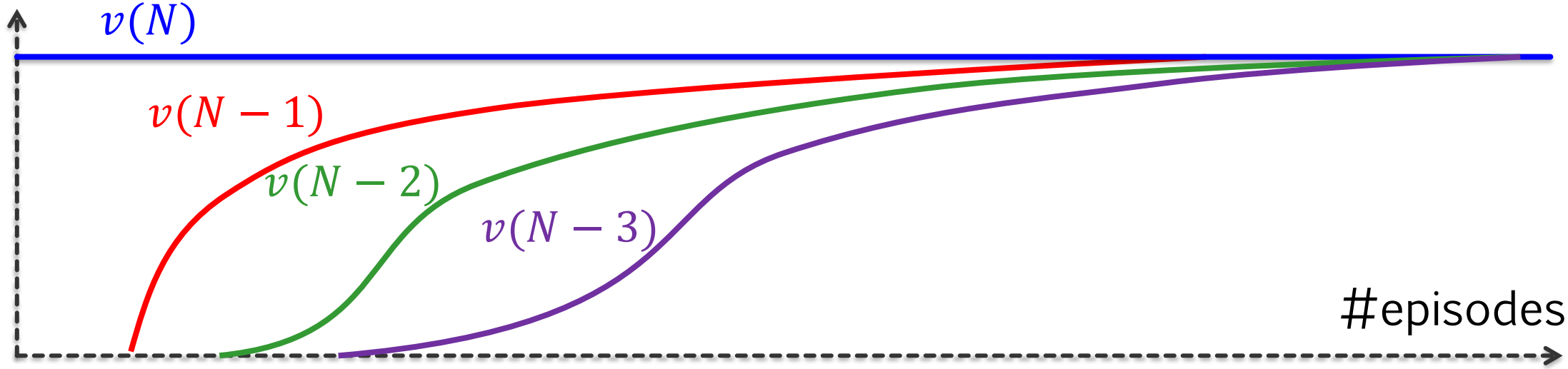
In TD(0), $v(3)$ is the target for $v(2)$, R_t is the target for $v(3)$

- $v(3) = 1$ right away
- $v(2)$ converges to 1 at a much slower rate (governed by α_n)

Let's conspire against TD(0)!



- MC: after one episode = N steps, all $v(s) = 1$ → done!
- TD: after one episode, $v(1) = v(2) = v(3) = \cdots = v(N-1) = 0,\ v(N) = 1$
  - After 2nd episode, $v(1) = v(2) = v(3) = \cdots = v(N-2) = 0,\ v(N-1) = \alpha$
  - After 3rd episode, $v(1) = v(2) = v(3) = \cdots = v(N-3) = 0,\ v(N-2) = \alpha^2, v(N-1) = 2\alpha(1-\alpha)$



From prediction to control

- Recall how we improved the policy based on current state-action value function estimates

$$\pi'(s) \coloneqq \arg\max_{a \in \mathcal{A}} q_\pi(s,a)$$

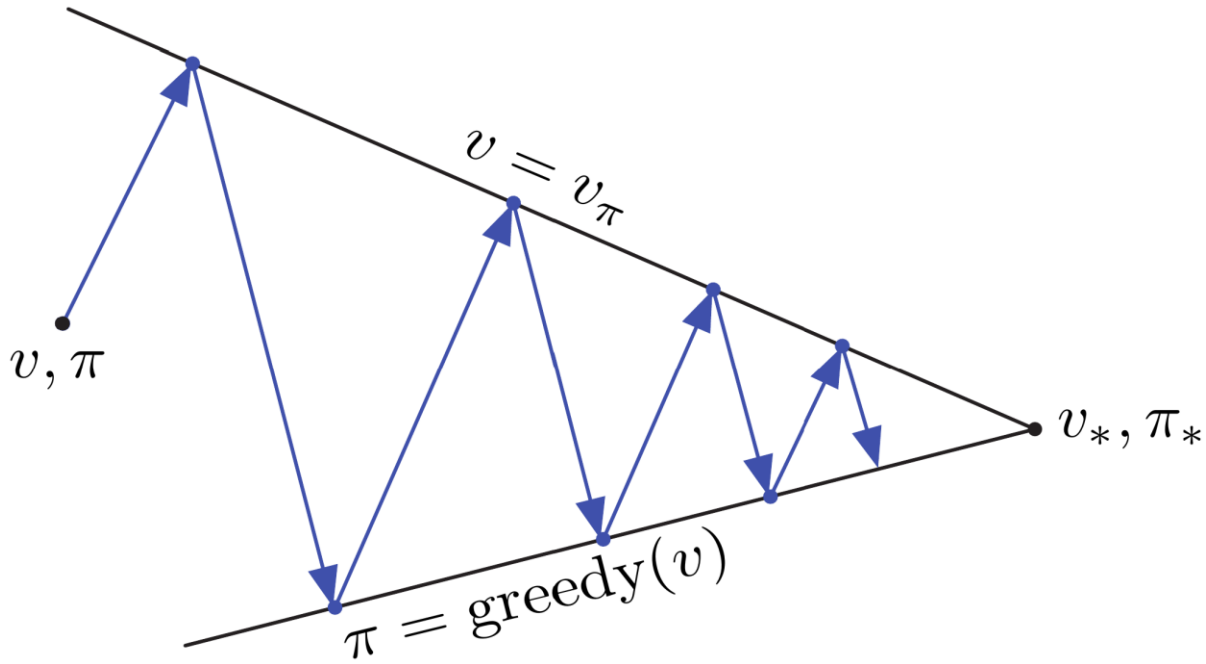
- In dynamic programming:

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

suffices to maintain only $v_\pi(s)$

- In Monte Carlo, we need to estimate

$$q(S_t, A_t) \coloneqq \text{average}(\text{returns}[(S_t, A_t)])$$



Generalized policy iteration

In TD-learning too, we need to estimate q

Estimating the state-action value function

Can we learn (estimate) $q_\pi(s,a)$ the same way we learned $v_\pi(s)$?

- For $v_\pi(s)$, for every transition $S_t, A_t, R_{t+1}, S_{t+1}$, we performed the update

$$v_\pi(S_t) \coloneqq v_\pi(S_t) + \alpha[R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)]$$

- Similarly, for $q_\pi(s,a)$, for every transition $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$, we update

$$q_\pi(S_t, A_t) \coloneqq q_\pi(S_t, A_t) + \alpha[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) - q_\pi(S_t, A_t)]$$

- Note: it is important that A_{t+1} is chosen according to π (on-policy)

Conceptually, the two updates are identical

- Both implement averaging of the target in an incremental way

Algorithm's name: SARSA, comes from the update based on $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

Using SARSA for policy optimisation

Just like in DP and MC methods, we can follow the greedy policy:

$$\pi'(s) \coloneqq \arg\max_{a \in \mathcal{A}} q_\pi(s, a)$$

To derive a good policy, the estimate of $q_\pi(s, a)$ must be accurate

- For accurate estimation, each state-action pair must be visited infinitely often
- May not happen with a deterministic policy
- We can do so by following an ϵ-greedy policy:

$$a_* \coloneqq \arg\max_a q(S_t, a)$$

$$\text{FOR } a \in \mathcal{A}\text{: } \pi(a|S_t) \coloneqq \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}| & \text{if } a = a_* \\ \epsilon/|\mathcal{A}| & if\ a \neq a_* \end{cases}$$

SARSA: on policy TD control

PARAMETERS: step-size $\alpha \in (0,1)$, small $\epsilon > 0$

INITIALIZE: $q(s,a)$ arbitrarily for all (s,a), except that $q(\otimes,\cdot) = 0$

REPEAT: for each episode

Pick initial state S_0

Pick action A_0 according to ϵ-greedy policy w.r.t. q

FOR $t = 1, 2, \ldots$:

Take action A_t, observe R_{t+1}, S_{t+1} from environment

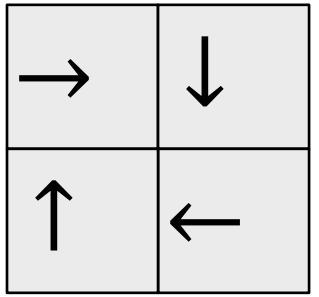
Pick action A_{t+1} according to ϵ-greedy policy π w.r.t. q

Update $q(S_t, A_t) \coloneqq q(S_t, A_t) + \alpha[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)]$

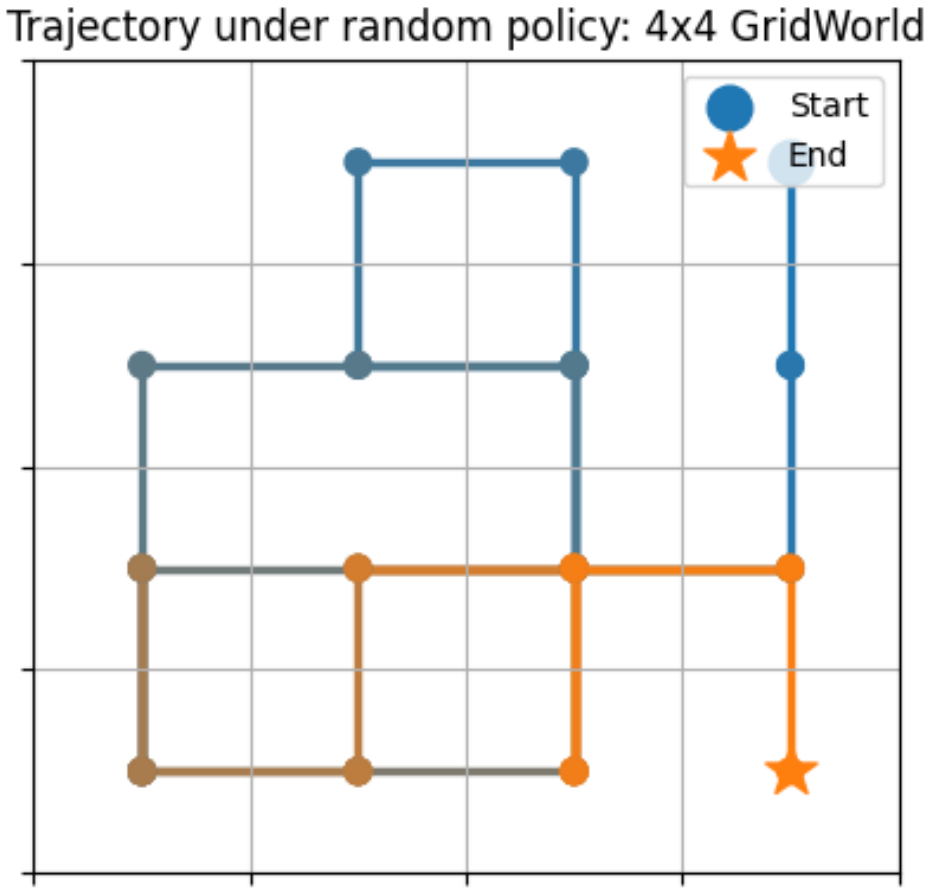
UNTIL end of episode $(S_t = \otimes)$

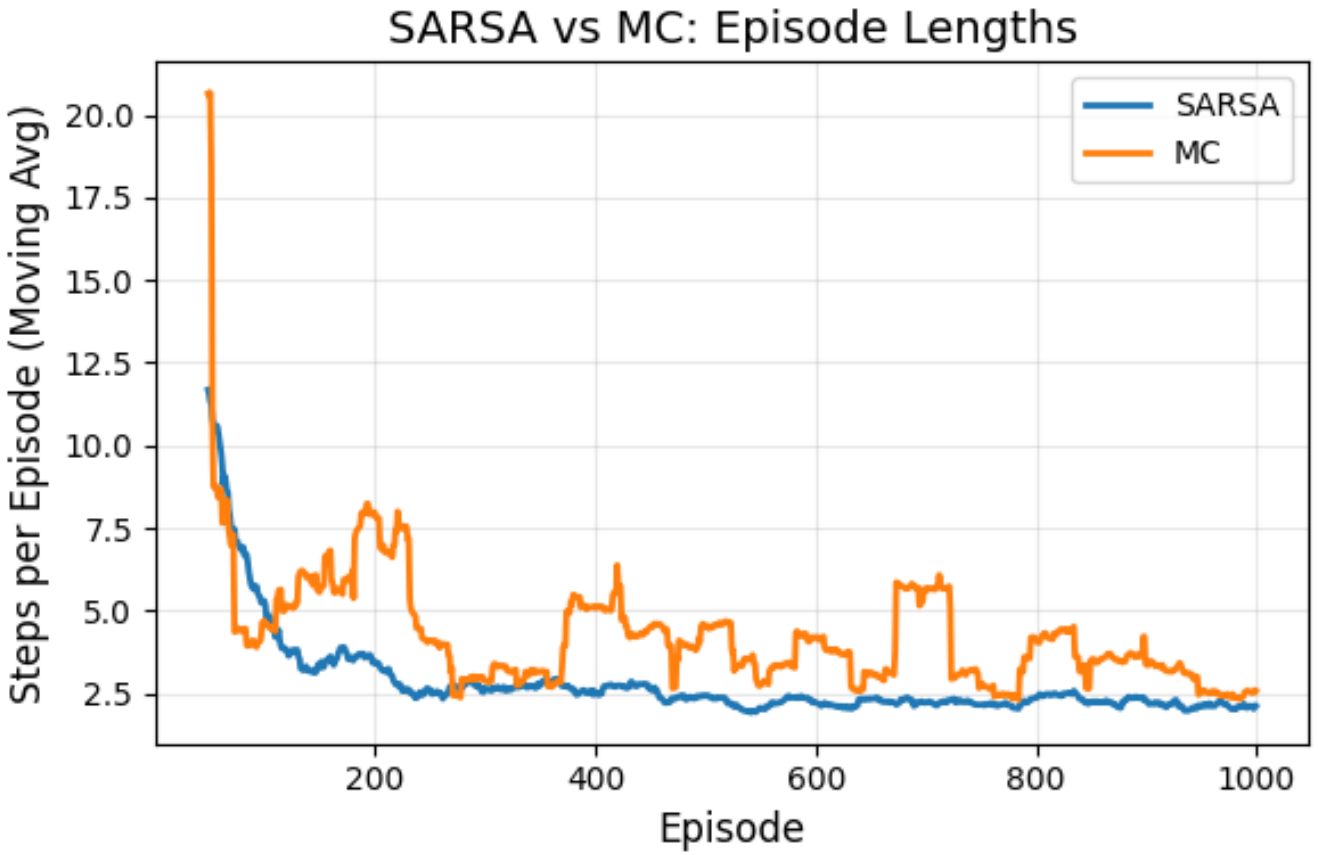
Some more observations on TD vs MC

- Handling continuing task:
  - TD can be used "out of the box"
  - MC is naturally episodic → samples of G_t, for continuing task need some artificial cut-off
    - E.g., when γ^n is lower than some threshold close to 0 → consider it a pseudo-episode, collect reward samples
- Loopy policy:
  - In some scenarios, MC may accidentally generate an intermediate policy π with a cycle
  - Under such a policy π, an episode may never terminate (if we hit one of the states in the cycle)
  - Therefore, the value function and policy π never gets updated → no convergence
  - In TD, this would only happen temporarily: value and therefore policy changes in every time step → no permanent blockage



TD v/s MC: small gridworld

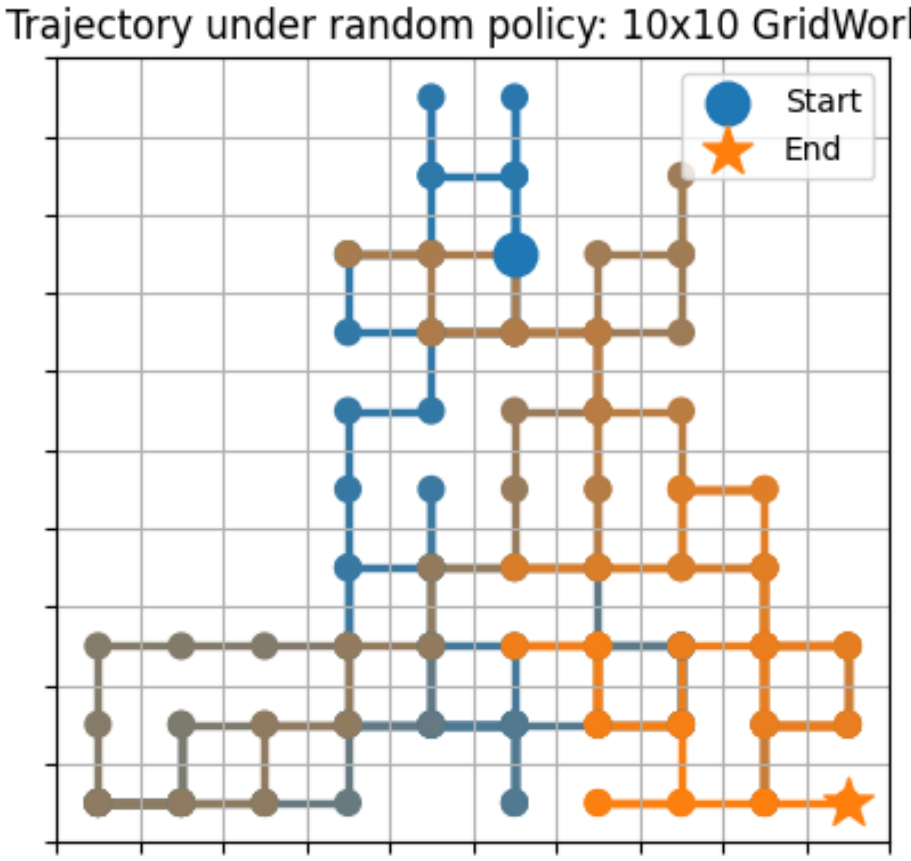


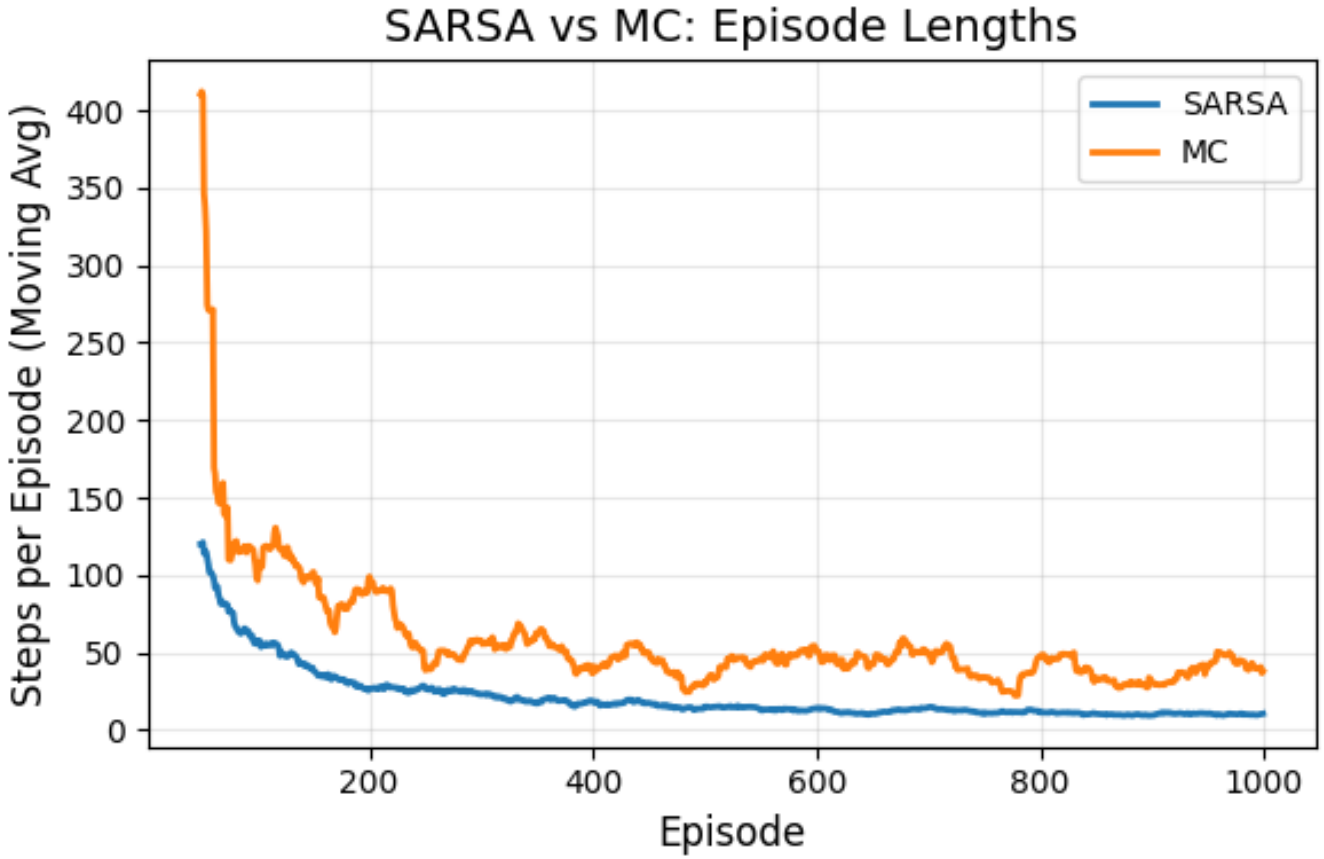


For a simple 4x4 gridworld, most episodes, starting from any state, with a policy taking random actions, converges within a few steps.

Therefore, performance of Monte Carlo and TD methods are comparable

TD v/s MC: large gridworld



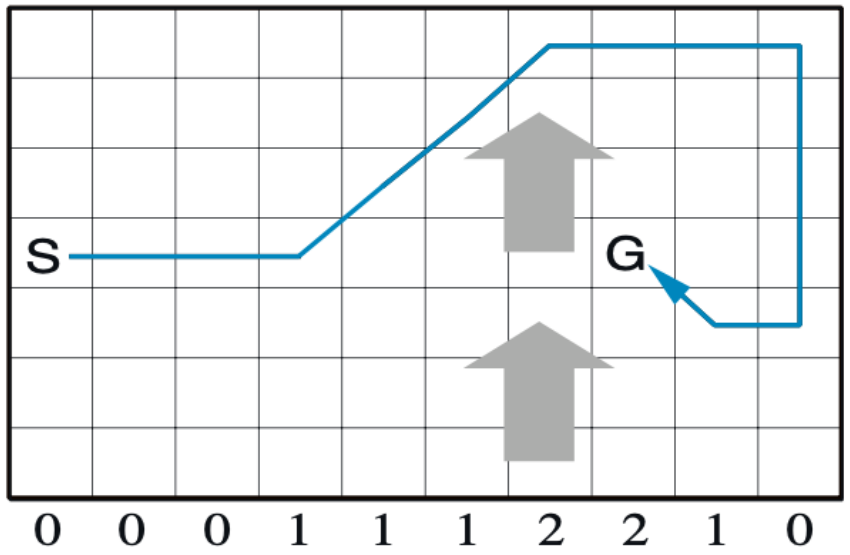


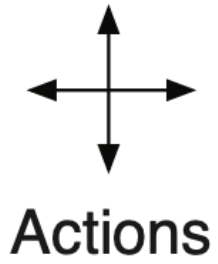
For a 10x10 gridworld, episodes take longer to converge.

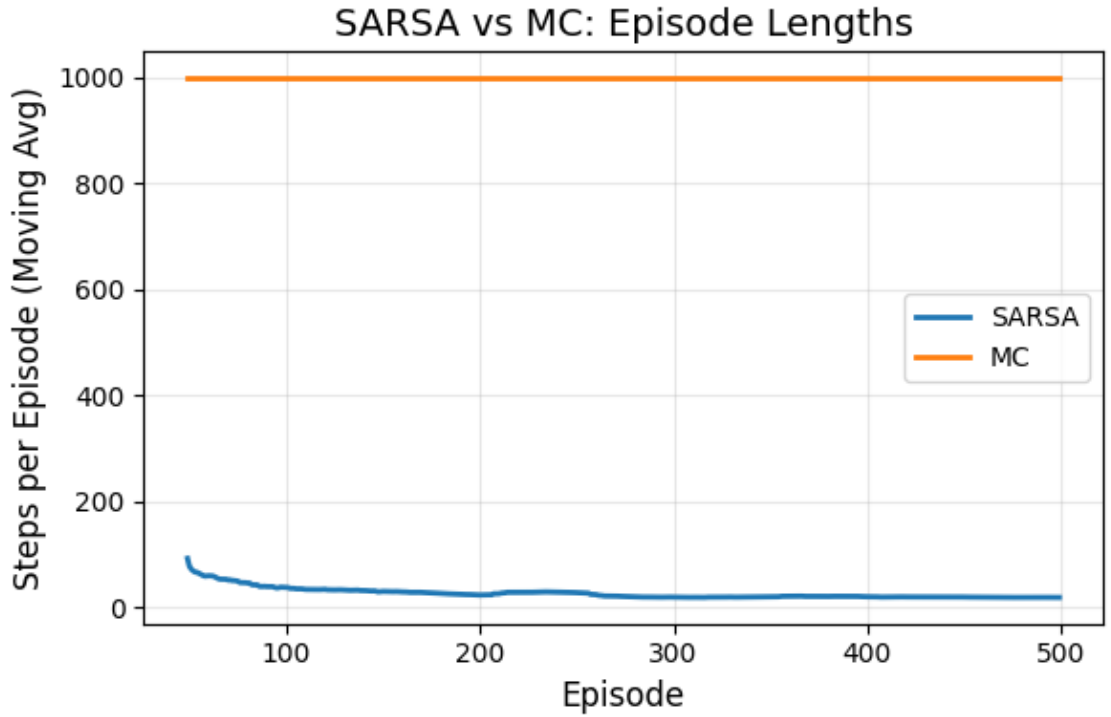
We see that TD learning is performing strictly better than Monte Carlo

TD v/s MC: windy gridworld

optimal trajectory







Consider a 7x10 gridworld, with a single start state (S) and a single end state (G)

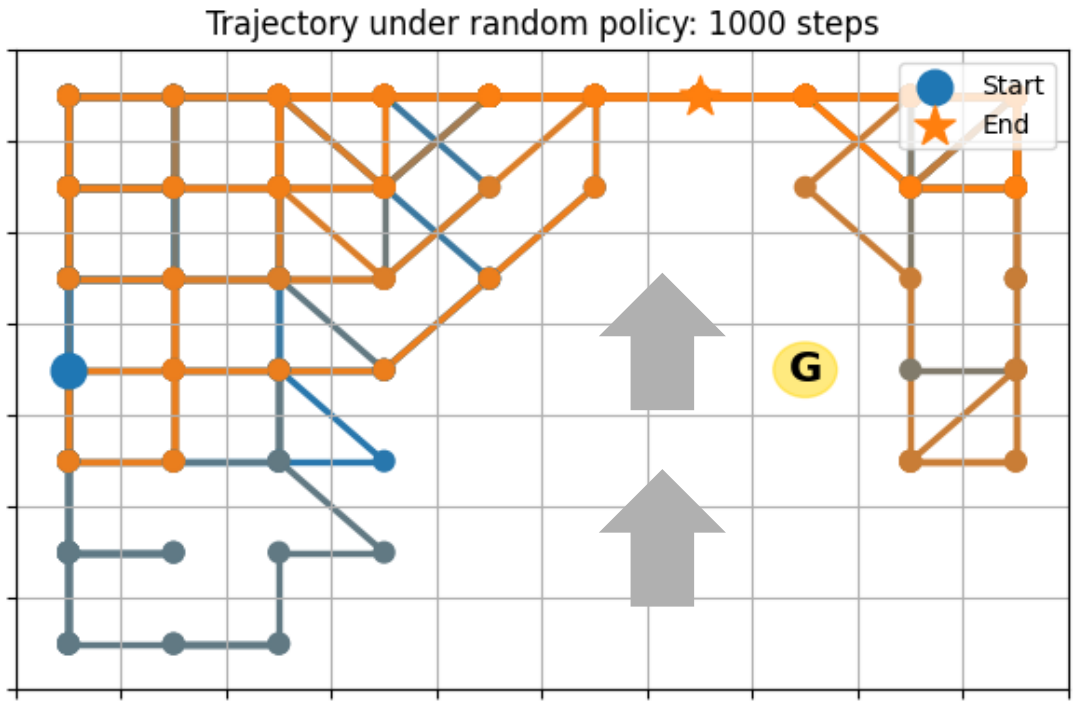
Standard actions: up, down, left, right.

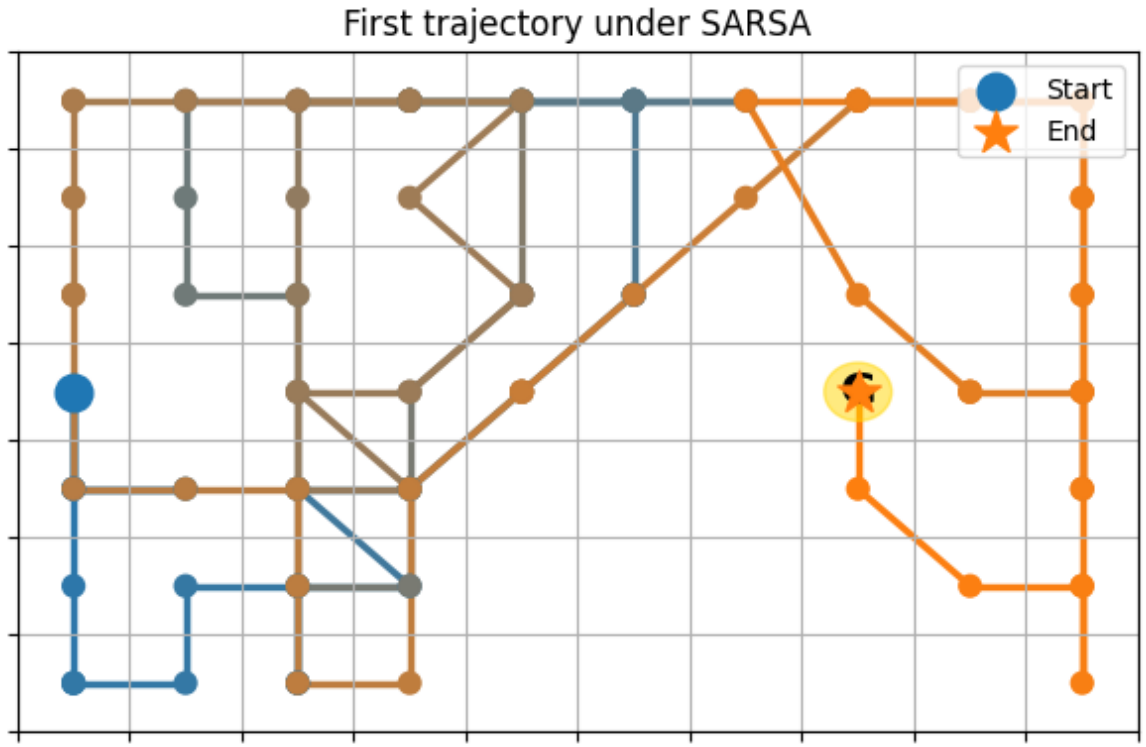
Upward blowing wind: moves the agent upward irrespective of action.

In this environment,

- Monte Carlo method is not able to learn anything,
- SARSA converges to optimal policy. Why?

TD v/s MC: windy gridworld





With random actions in windy world,

- agent likely to get stuck at top
- the episode doesn't end even after 1000 steps!
- no signal for Monte Carlo to learn

In SARSA,

- agent learns some estimate of $q(s,a)$ during the first episode itself
- learns: → at top row, ↓ from right col.
- eventually reaches target

Q-learning: off-policy TD(0)

- Recall SARSA: $q_\pi(S_t, A_t) \coloneqq q_\pi(S_t, A_t) + \alpha[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) - q_\pi(S_t, A_t)]$ where we constrain π to be ϵ-soft
  - ϵ controls the amount of exploration
- Q-learning: $q(S_t, A_t) \coloneqq q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)\right]$
  - Off-policy: we follow some policy π, but learn q that approximates q_*
  - π drives exploration, but quality function q learns about optimal policy
- Convergence guaranteed as long as all (s, a) are visited asymptotically infinitely often
  - Other than that, actual policy π does not influence learned q-function – which converges to q_*

Q-learning: off-policy TD control

PARAMETERS: step-size $\alpha \in (0,1]$, small $\epsilon > 0$

INITIALIZE: $q(s,a)$ arbitrarily for all (s,a), except that $q(\otimes,\cdot) = 0$

REPEAT: for each episode

Pick initial state S_0

Pick action A_0 according to ϵ-greedy policy w.r.t. q

FOR $t = 0, 1, 2, \ldots$:

Pick action A_{t+1} according to π

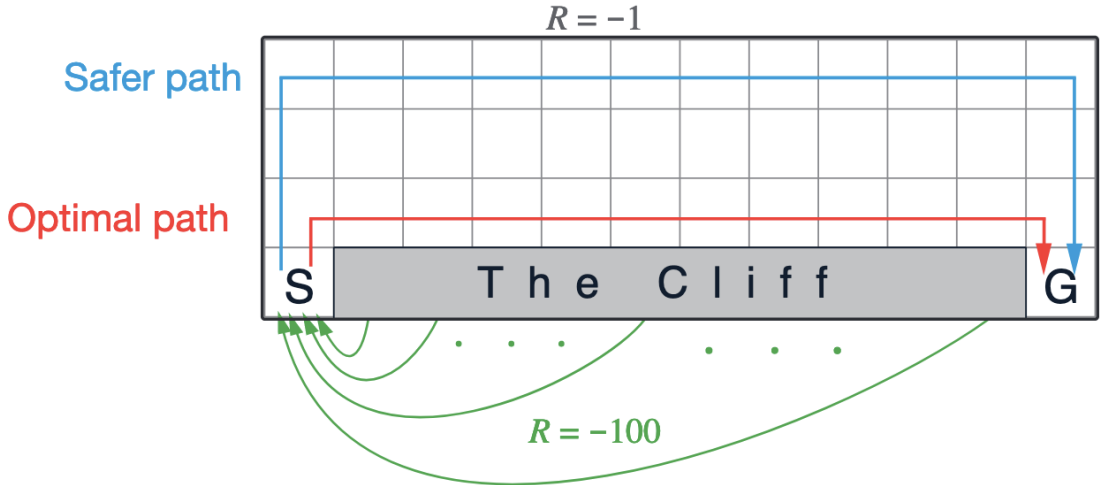
Take action A_t, observe R_{t+1}, S_{t+1}

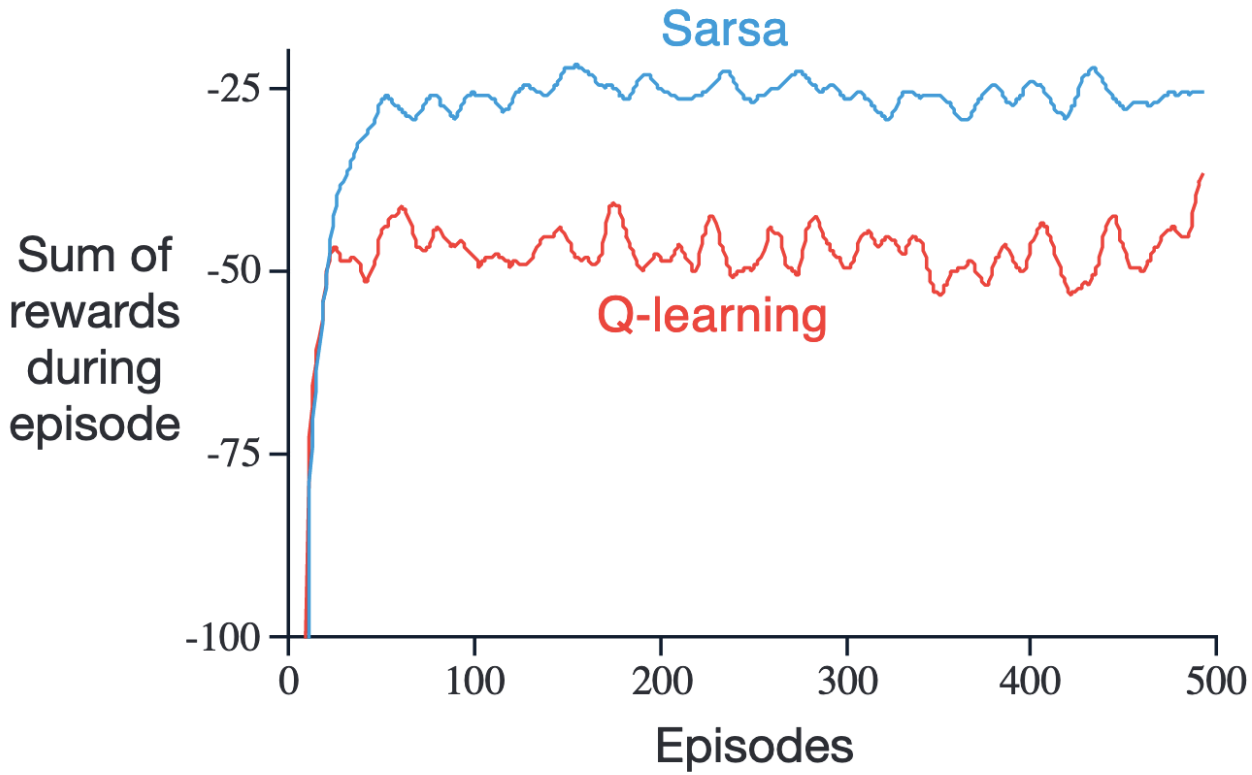
Update $q(S_t, A_t) \coloneqq q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)]$

UNTIL end of episode ($S_t = \otimes$)

Q-learning v/s SARSA

- Cliffwalk: standard undiscounted episodic task:
  - Walk quickly from S to G
  - Don't fall off the cliff
- 0.1-greedy action selection for both SARSA and Q-learning
- SARSA learns the optimal policy within the ϵ-soft class: stay away from the cliff, because every 10th step is random
- Q-learning learns the optimal policy π_*: walk right along the cliff
  - This does less well under the actual ϵ-soft policy
  - Note: π_* has $G_0 = -13$





Summary

- MC: for every s or (s, a) pair, average all rewards after first visit until termination
  - Implicitly: every future reward is equally important
- TD(0): Bellman equations say that value estimate of s or (s, a) should be close to (next reward + estimated value of next state)
  - Move estimator towards this target value
  - Implicitly: new reward taken into account immediately, the future is bootstrapped from the value of the next state
  - TD usually beats MC, but not uniformly (counterexample: long one-directional chain)
- Robbins-Monro conditions for stochastic approximation
  - Decay not too fast and not too slow
- SARSA: on-policy learning of optimal policy
  - Exploration controlled via ϵ-soft π
- Q-learning: off-policy learning of optimal policy
  - Behavioral policy drives exploration directly, method learns π_* regardless