Temporal Difference Learning

Chris Amato
Northeastern University

with some slides from Rob Platt, Lawson Wong and U Alberta

"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."

– SB, Ch 6

Announcements

- MC assignment due **10/19**
- Exam 10/20
- Project proposal due 10/24
- Read chapter 6 (TD)
- Read 7-7.3 (n-step returns)

Exam

- Covers through MC (not these slides)
- Multiple choice, short answer & homework style questions
- Some other resources for practice:
  - David Silver's course
    - http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
  - The Berkeley AI course
    - http://ai.berkeley.edu/exams.html

Project

- An excuse to learn about new RL methods or apply current RL methods to a problem you care about
- High-quality research paper (something novel/interesting)
- 1 or 2 people (each with a different algorithm)
- Examples:
  - Recreate algorithm and results from a paper (with analysis)
  - Compare and analyze some algorithms we studied on a problem of your choice
  - Implement an algorithm, run and analyze on a new domain
  - Improve an algorithm and compare with original version

Project

- Project proposal (one or two pages) due 10/24
  - This requires some research, so start soon!
- Project presentations 12/1 and 12/5
- Project report due 12/11
- Primarily graded on presentation and report, but upload code etc.
- Two most common notes on proposals and papers:
  - More formal
  - More analysis

Temporal-difference learning

"If one had to identify one idea as
central and novel to reinforcement learning,
it would undoubtedly be temporal-difference (TD) learning."

Our MDP journey

0. Markov decision process $p(s', r \mid s, a)$

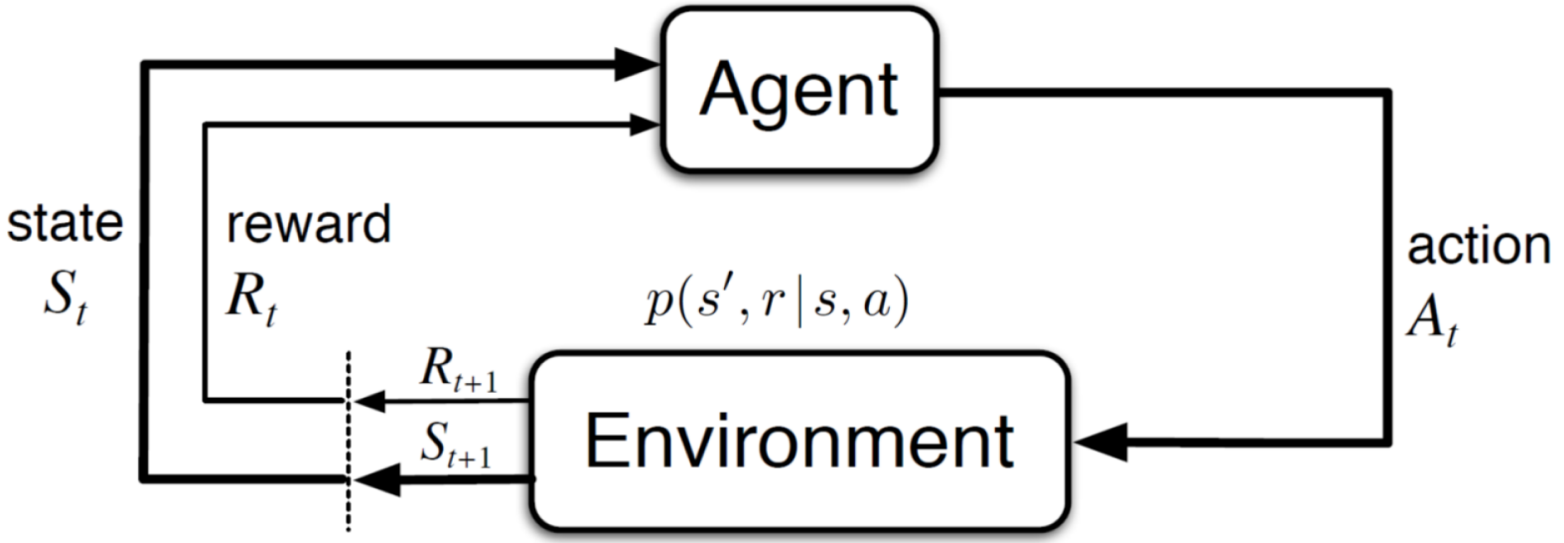


Figure 3.1: The agent–environment interaction in a Markov decision process.

$$\text{MDP } \mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$$

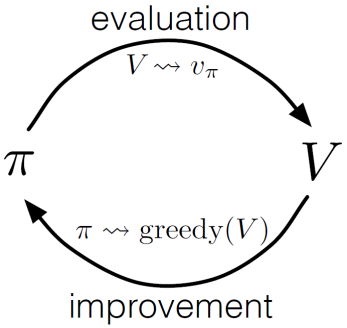
$=$ (state space, action space, transition fn., reward fn., discount factor)

Transition / reward functions shown as $p(s', r|s, a)$ above

Our MDP journey

0. Markov decision process $p(s', r \mid s, a)$

1. Dynamic programming



- Value iteration

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

- Policy evaluation

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

- Policy iteration

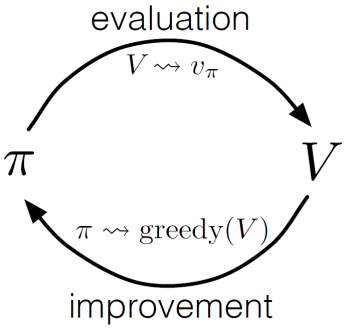
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

where $\xrightarrow{\text{E}}$ denotes a policy *evaluation* and $\xrightarrow{\text{I}}$ denotes a policy *improvement*

Our MDP journey

0. Markov decision process $p(s', r \mid s, a)$

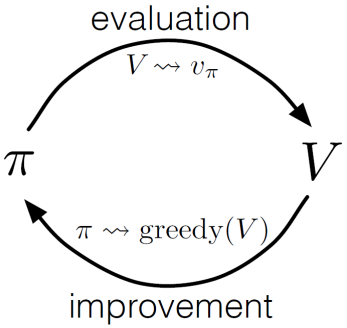
1. Dynamic programming
   Value iteration, policy evaluation, policy iteration



$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$$

Our MDP journey

0. Markov decision process *p(s', r | s, a)*



1. Dynamic programming
   Value iteration, policy evaluation, policy iteration

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

2. Monte-Carlo methods (cf. Multi-armed bandits)

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

$$V_\pi(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} \left[G_t | S_t = s \right]$$

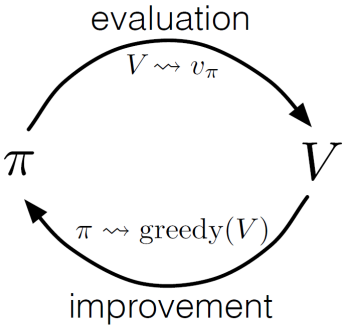
$$= \frac{1}{N(s)} \sum_{i=1}^{N(s)} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}^{(i)} \middle| S_t = s \right]$$

$$= \frac{1}{N(s)} \sum_{i=1}^{N(s)} \left[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \middle| S_t = s \right]$$

Sample average of episodic returns

Our MDP journey

0. Markov decision process *p(s', r | s, a)*



1. Dynamic programming
   Value iteration, policy evaluation, policy iteration

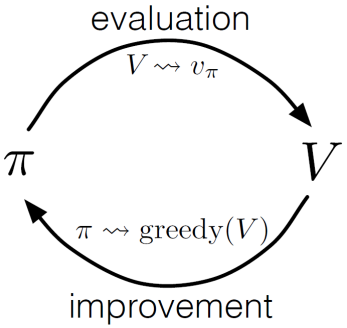
$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

2. Monte-Carlo methods (cf. Multi-armed bandits)
   Sample average of episodic returns

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

Where G_t denotes total return
after the (first) visit to s_t

Our MDP journey



0. Markov decision process *p(s', r | s, a)*

1. Dynamic programming
   Value iteration, policy evaluation, policy iteration

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

2. Monte-Carlo methods (cf. Multi-armed bandits)
   Sample average of episodic returns

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

3. Temporal-difference learning

Temporal-difference learning

"If one had to identify one idea as
central and novel to reinforcement learning,
it would undoubtedly be temporal-difference (TD) learning."

"TD learning is a combination of Monte-Carlo ideas
and dynamic programming (DP) ideas."

Temporal-difference learning

"If one had to identify one idea as
central and novel to reinforcement learning,
it would undoubtedly be temporal-difference (TD) learning."

"TD learning is a combination of Monte-Carlo ideas
and dynamic programming (DP) ideas."

"Like Monte-Carlo methods, TD methods can
learn directly from raw experience
without a model of the environment's dynamics."

Temporal-difference learning

"If one had to identify one idea as
central and novel to reinforcement learning,
it would undoubtedly be temporal-difference (TD) learning."

"TD learning is a combination of Monte-Carlo ideas
and dynamic programming (DP) ideas."

"Like Monte-Carlo methods, TD methods can
learn directly from raw experience
without a model of the environment's dynamics."

"Like DP, TD methods update estimates
based in part on other learned estimates,
without waiting for a final outcome (they bootstrap)."

DP vs. MC vs. TD

Dynamic programming:

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

Monte Carlo:

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

Temporal difference:

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

DP vs. MC vs. TD

Dynamic programming:

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

Monte Carlo:

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

Temporal difference:

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

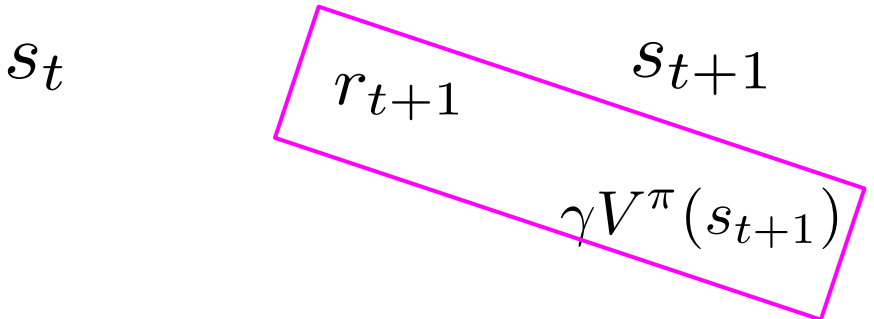
Temporal Difference Learning

Monte Carlo:

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

TD Learning:

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



Temporal Difference Learning

Monte Carlo:

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

TD Learning:

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

TD Error == δ_t

DP vs. MC vs. TD

Dynamic programming:

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma V(s') \Big]$$

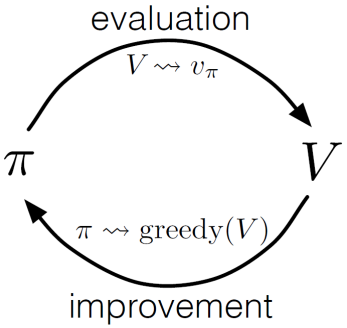
Monte Carlo:

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

Temporal difference:

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

Our MDP journey



0. Markov decision process *p(s', r | s, a)*

1. Dynamic programming
   Value iteration, policy evaluation, policy iteration

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

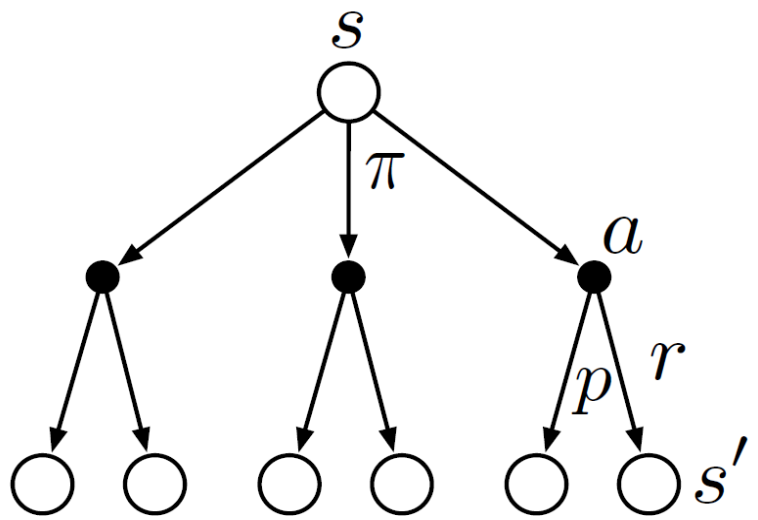
2. Monte-Carlo methods (cf. Multi-armed bandits)
   Sample average of episodic returns

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

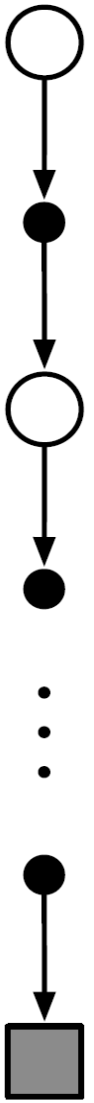
3. Temporal-difference learning
   Sample average of <u>one-step returns</u>

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

DP vs. MC vs. TD

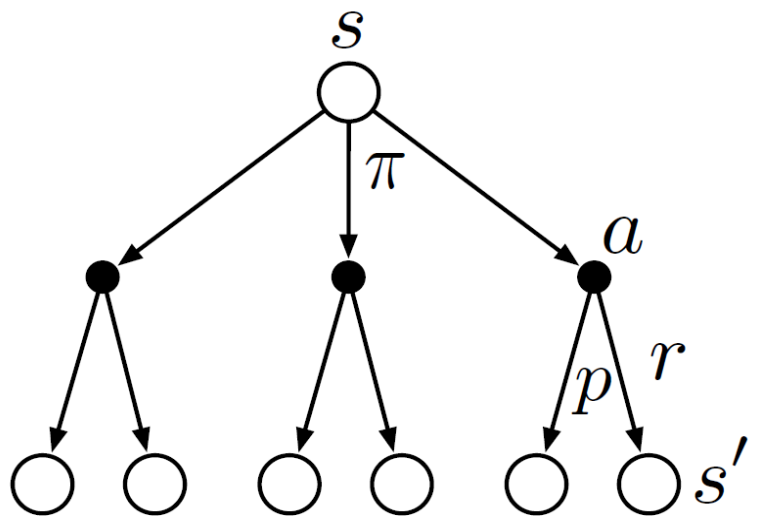


Backup diagram for v_π

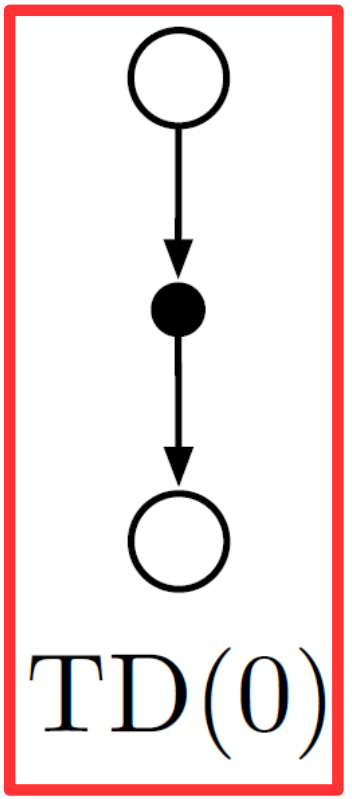


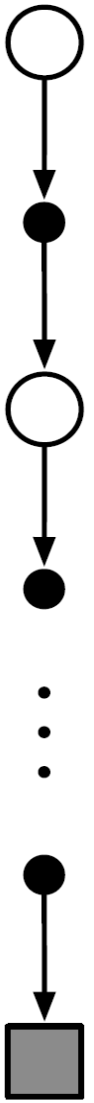
Dynamic programming
Monte-Carlo

DP vs. MC vs. TD



Backup diagram for v_π





Dynamic programming
Monte-Carlo

What issue does TD resolve?

Dynamic programming:

$$V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$$

What issue does TD resolve?

Dynamic programming:

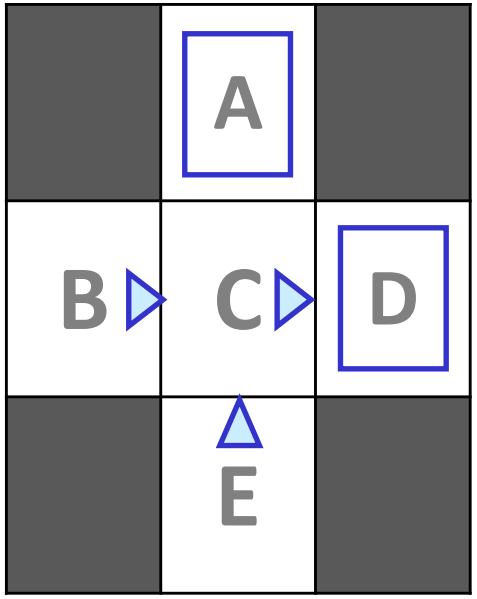
$$V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$$

Monte Carlo?

Monte-carlo estimates example

States: A,B,C,D,E
Actions: left, right, up, down, exit

0.2 movement noise in C only



Blue arrows denote policy that agent is following during learning (exploration policy)

Observations (from 4 episodes): (s, a, s', r) tuples

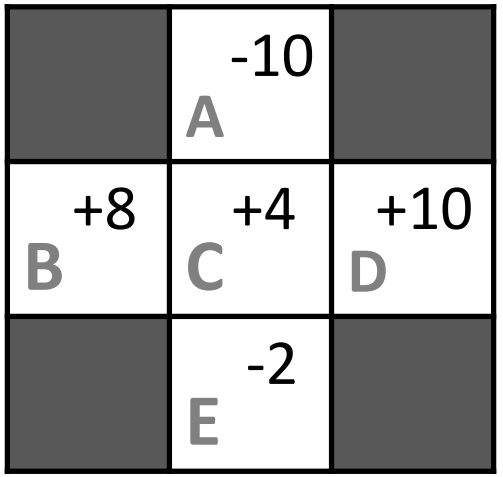
1. B, right, C, -1
   C, right, D, -1
   D, exit, term, 10

2. E, up, C, -1
   C, right, A, -1
   A, exit, term, -10

3. B, right, C, -1
   C, right, D, -1
   D, exit, term, 10

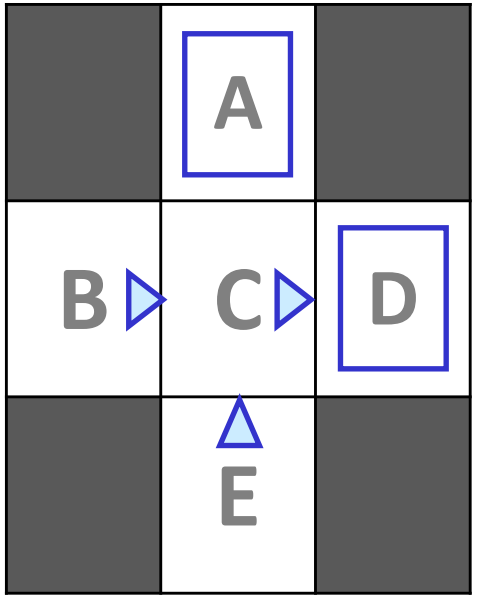
4. E, up, C, -1
   C, right, D, -1
   D, exit, term, 10

Estimates:
Estimate V(s) as the sample average of returns when starting from s



Monte-carlo estimates example

States: A,B,C,D,E
Actions: left, right, up, down, exit



Blue arrows denote policy that agent is following during learning (exploration policy)

Observations (from 4 episodes): (s, a, s', r) tuples

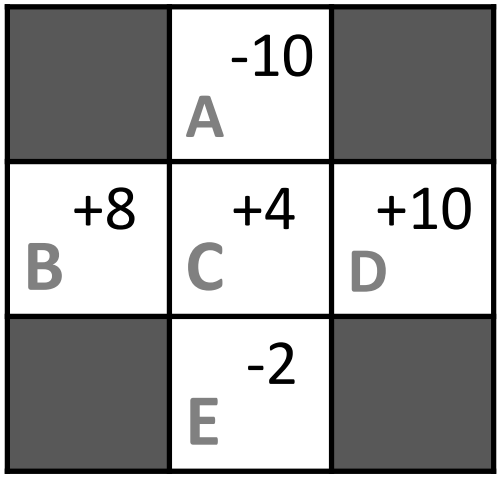
1. B, right, C, -1
   C, right, D, -1
   D, exit, term, 10

2. E, up, C, -1
   C, right, A, -1
   A, exit, term, -10

3. B, right, C, -1
   C, right, D, -1
   D, exit, term, 10

4. E, up, C, -1
   C, right, D, -1
   D, exit, term, 10

Estimates:
Estimate V(s) as the sample average of returns when starting from s



… but B and E are basically identical (both can only lead to C)

What issue does TD resolve?

Values of adjacent states are closely related:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

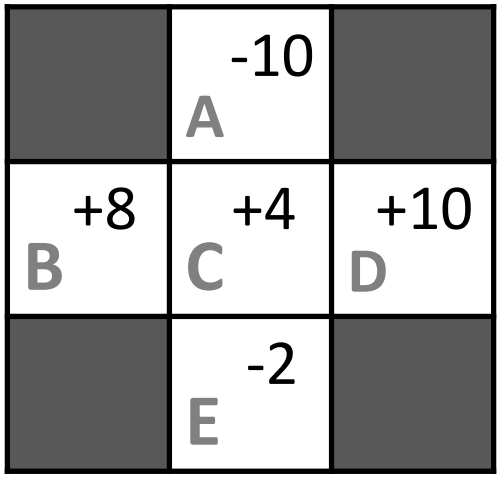
What issue does TD resolve?

Values of adjacent states are closely related:

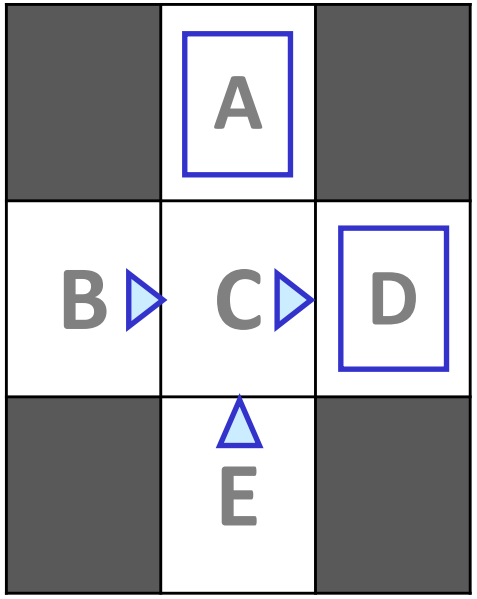
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

In this example,
if V(C) = +4,
then V(B) = -1 + 1 * (+4) = 3
V(E) = -1 + 1 * (+4) = 3

(each non-terminal state
has reward -1;
P(C | B, right) = 1
P(C | E, up) = 1)



What issue does TD resolve?



V(s)	A	B	C	D	E
Values for <u>known</u> MDP:	-10	4	5	10	4
Values for <u>Monte-Carlo</u> RL:	-10	3.89	4.79	10	3.69
Values for <u>temporal-difference</u> RL:	-10	4.00	4.87	10	3.98

$\alpha = 0.5$, 10^4 episodes

SB Example 6.1: Driving home

Scenario: you are leaving work to drive 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

SB Example 6.1: 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

Initial estimate

SB Example 6.1: 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

Add 10 min b/c of rain on highway

SB Example 6.1: 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

Subtract 5 min b/c highway was faster than expected

SB Example 6.1: 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

Behind truck, add 5 min

SB Example 6.1: Driving home

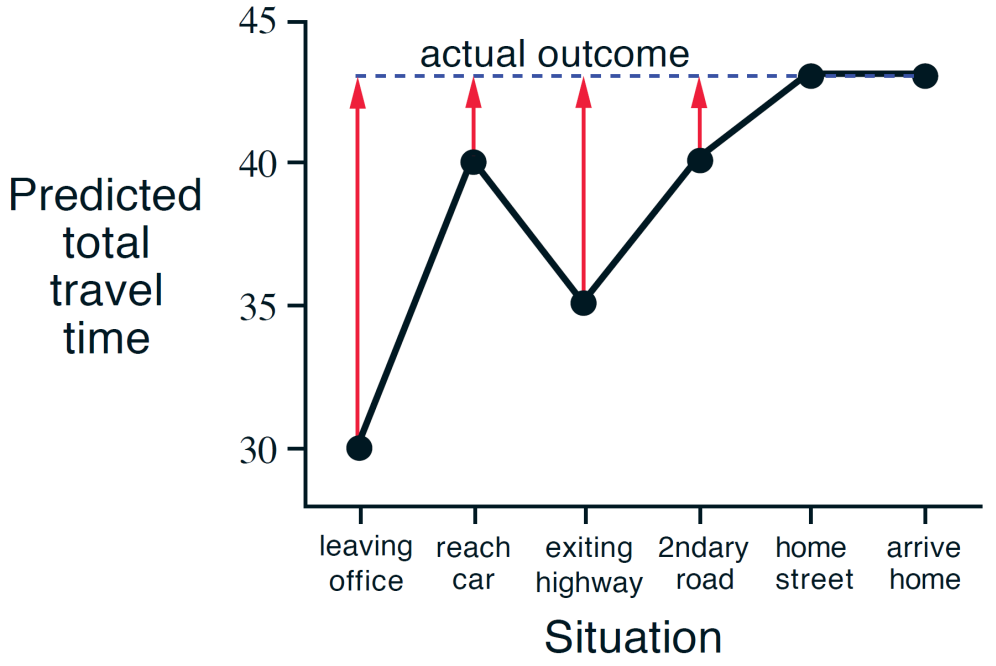
State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Almost home

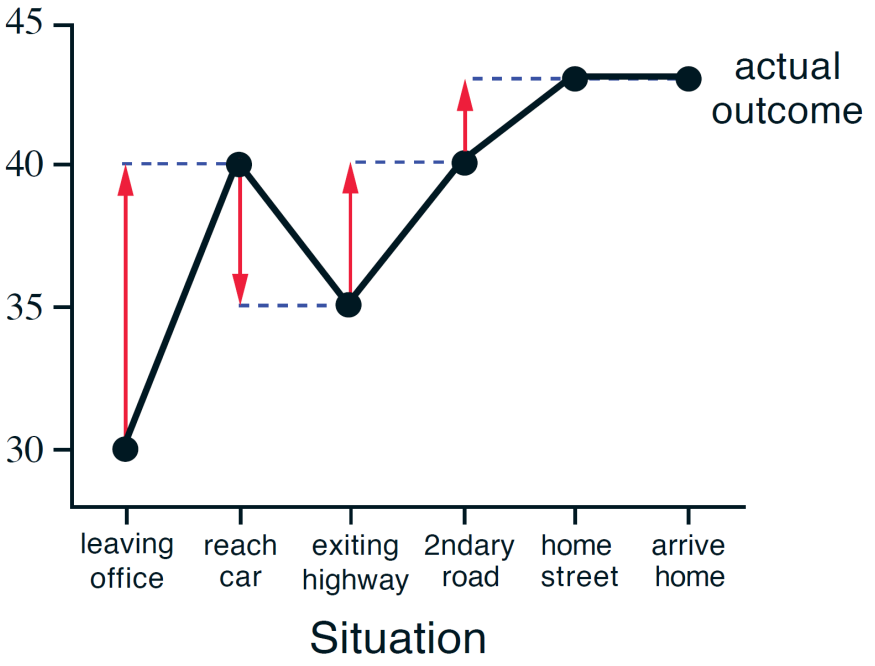
SB Example 6.1: Driving home

Suppose we want to estimate average time-to-go from each point along journey...

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



MC updates



TD updates

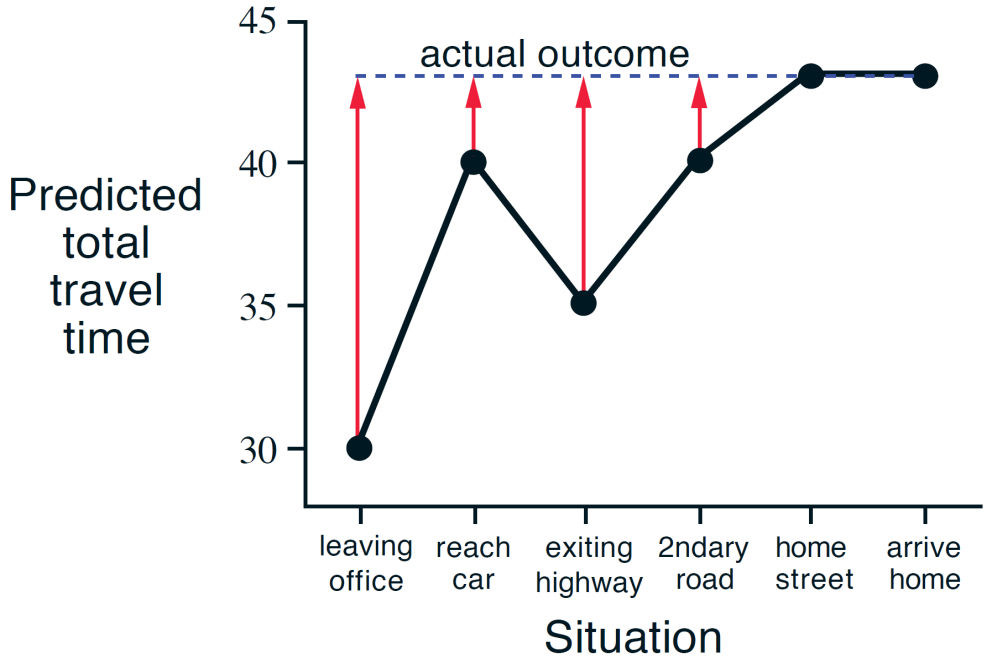
SB Example 6.1: Driving home

Suppose we want to estimate average time-to-go from each point along journey...

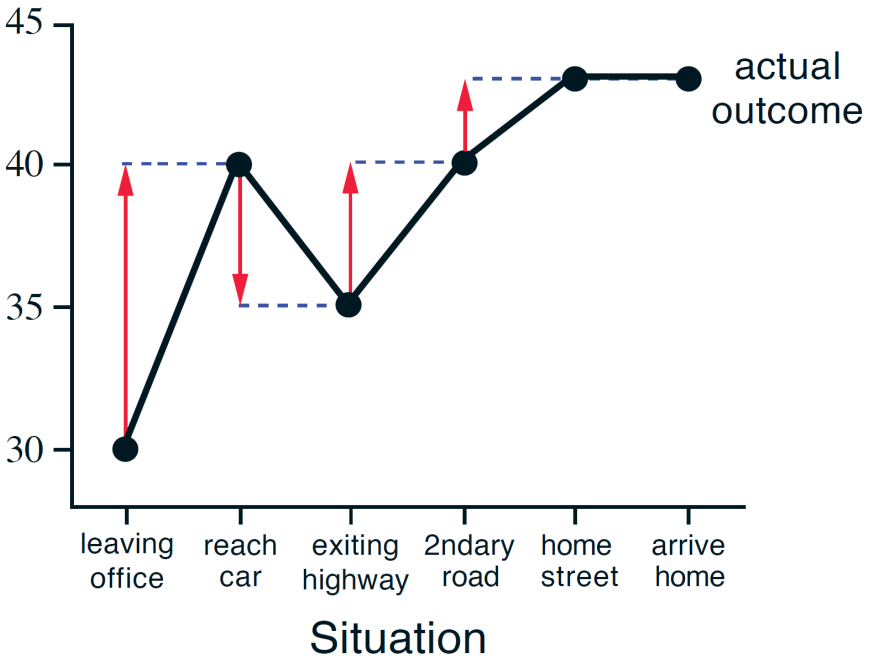
MC waits until the end before updating estimate

2ndary	nd truck	30
entering	reet	40
arrive home		43

redicted me to Go	*Predicted Total Time*
30	30
35	40
15	35
10	40
3	43
0	43



MC updates



TD updates

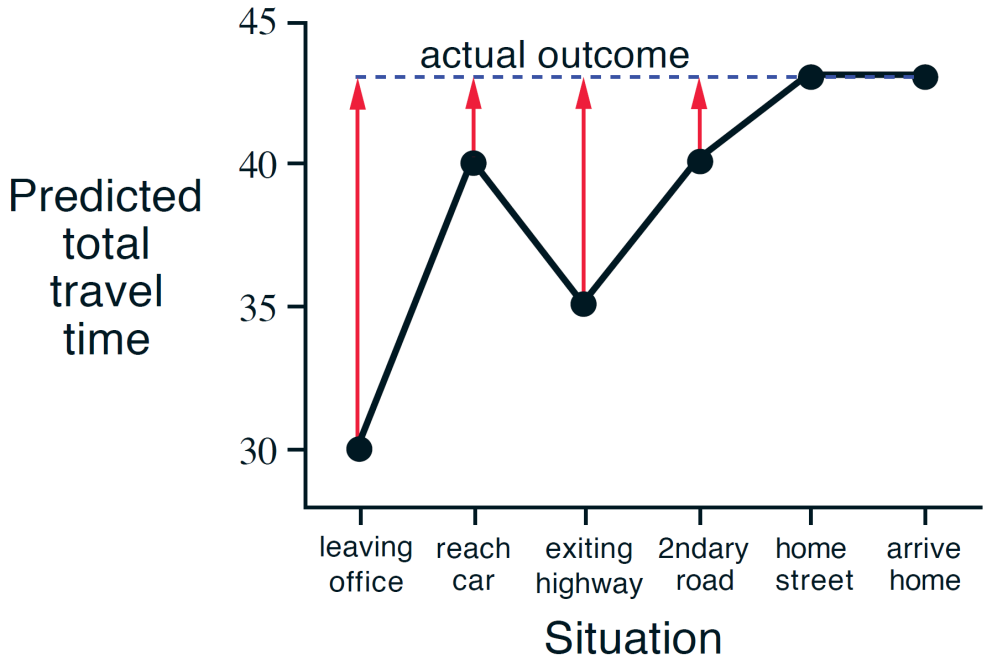
SB Example 6.1: Driving home

Suppose we want to estimate average time-to-go from each point along journey...

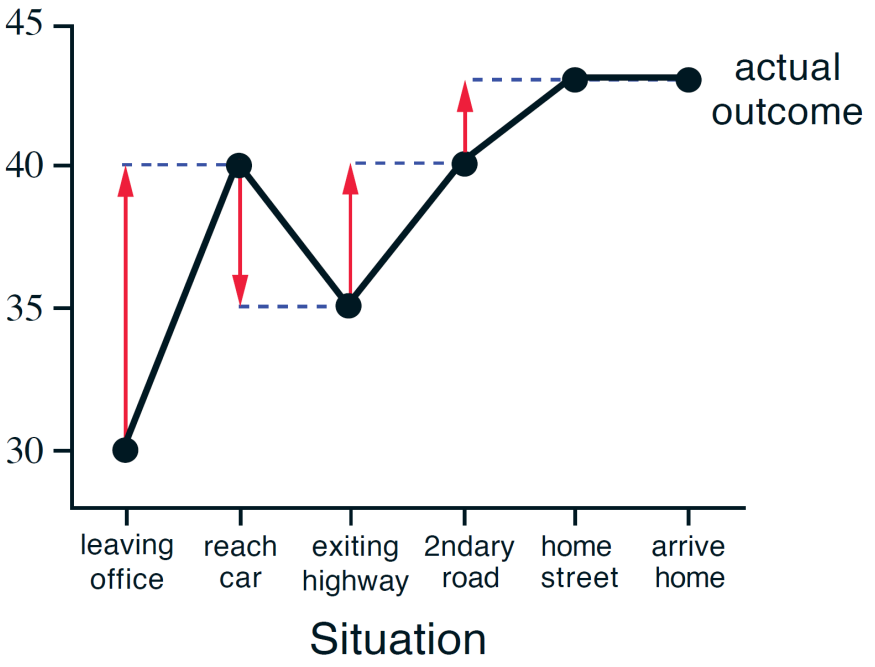
State	*Elapsed (minu*
leaving office, friday at 6	0
reach car, raining	5
exiting highway	20
2ndary road, behind truck	30
entering home street	40
arrive home	43

TD updates estimate as it goes

10	40
3	43
0	43



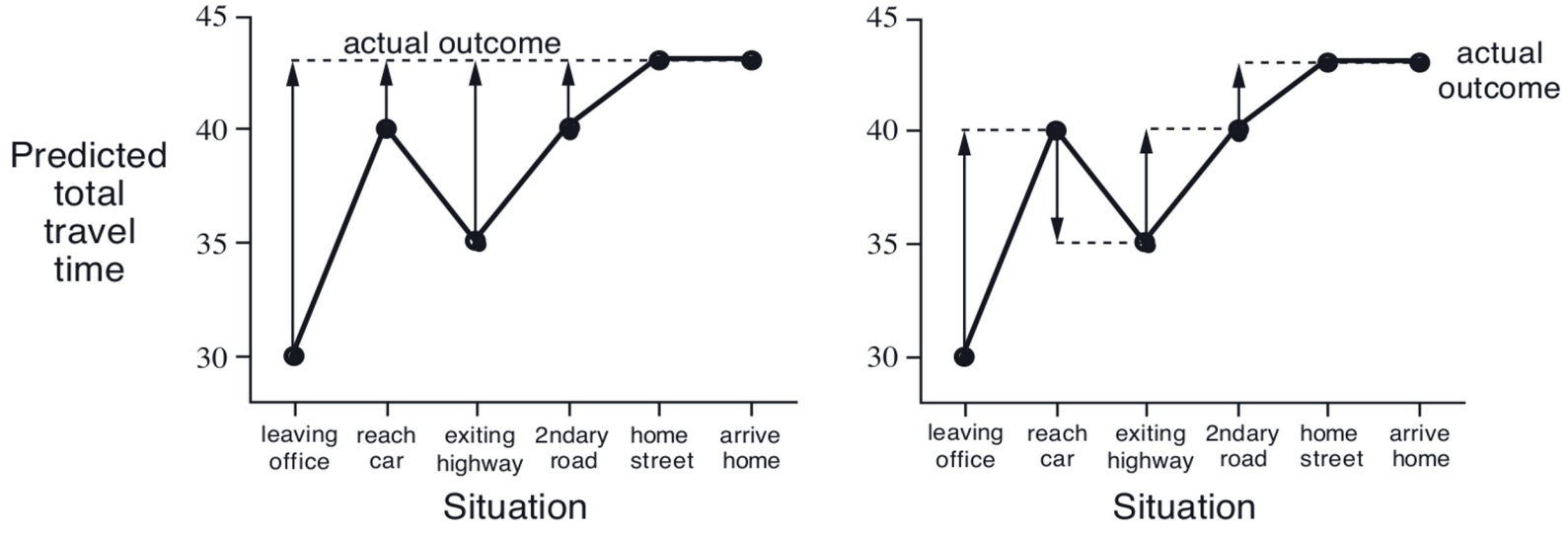
MC updates



TD updates

Think-pair-share

Exercise 6.2 This is an exercise to help develop your intuition about why TD methods are often more efficient than Monte Carlo methods. Consider the driving home example and how it is addressed by TD and Monte Carlo methods. Can you imagine a scenario in which a TD update would be better on average than a Monte Carlo update? Give an example scenario—a description of past experience and a current state—in which you would expect the TD update to be better. Here's a hint: Suppose you have lots of experience driving home from work. Then you move to a new building and a new parking lot (but you still enter the highway at the same place). Now you are starting to learn predictions for the new building. Can you see why TD updates are likely to be much better, at least initially, in this case? Might the same sort of thing happen in the original scenario? □



MC updates **TD updates**

Temporal Difference Learning

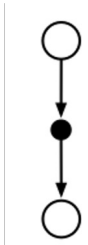
TD(0) for ***estimating*** V^{π}:

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^{+}$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
    until S is terminal

Backup Diagrams

SB represents various different RL update equations pictorially as *Backup Diagrams:*

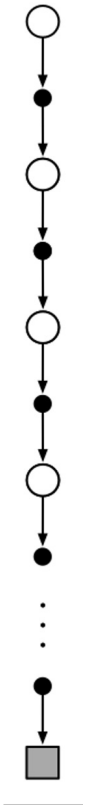
TD



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

– Why is the TD backup diagram short?
– Why is the MC diagram long?

MC



$$V^{\pi}(s_t) \leftarrow (1-\alpha)V^{\pi}(s_t) + \alpha G_t$$

Backup Diagrams

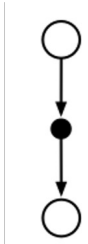
SB represents various different RL update equations pictorially as *Backup Diagrams:*

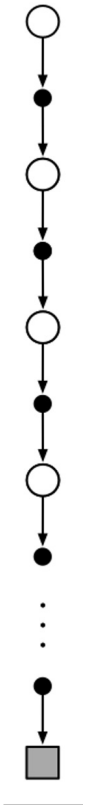
TD

State

State-action pair

MC



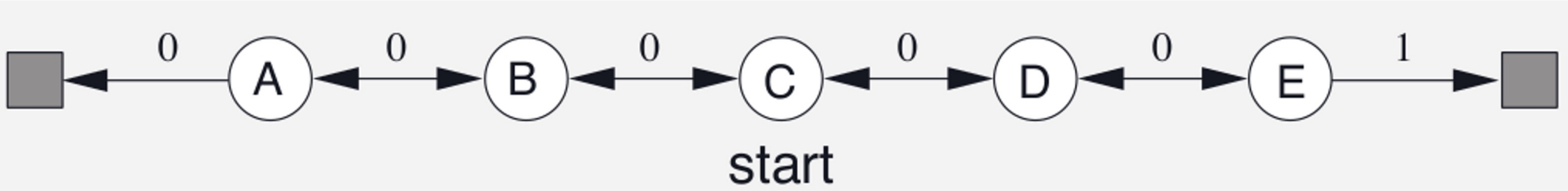


$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha[r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)]$$

– Why is the TD backup diagram short?
– Why is the MC diagram long?

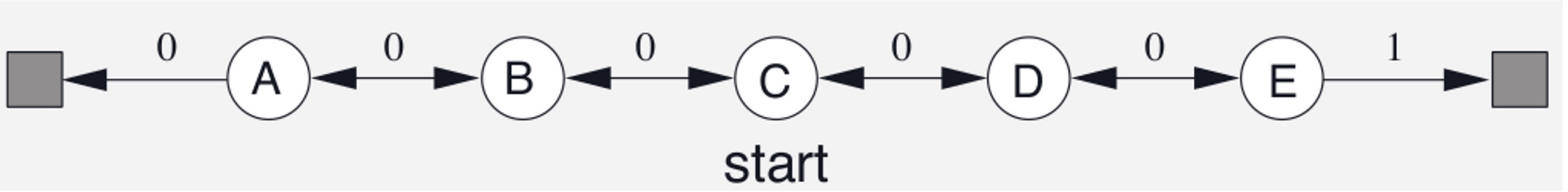
$$V^{\pi}(s_t) \leftarrow (1-\alpha)V^{\pi}(s_t) + \alpha G_t$$

SB Example 6.2: Random Walk



– This is a Markov Reward Process (MDP with no actions)
– Episodes start in state C
– On each time step, there is an equal probability of a left or right transition
– +1 reward at the far right, 0 reward elsewhere
– discount factor of 1
– the true values of the states are: $V(A) = \frac{1}{6}, V(B) = \frac{2}{6}, V(C) = \frac{3}{6}, V(D) = \frac{4}{6}, V(E) = \frac{5}{6},$

Think-pair-share

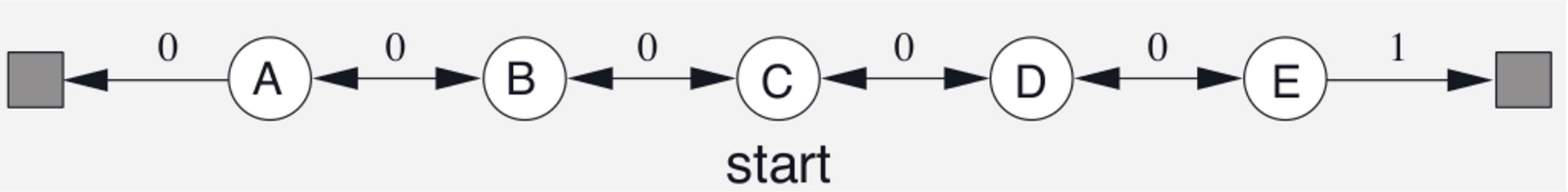


– This is a Markov Reward Process (MDP with no actions)
– Episodes start in state C
– On each time step, there is an equal probability of a left or right transition
– +1 reward at the far right, 0 reward elsewhere
– discount factor of 1
– the true values of the states are: $V(A) = \frac{1}{6}, V(B) = \frac{2}{6}, V(C) = \frac{3}{6}, V(D) = \frac{4}{6}, V(E) = \frac{5}{6},$

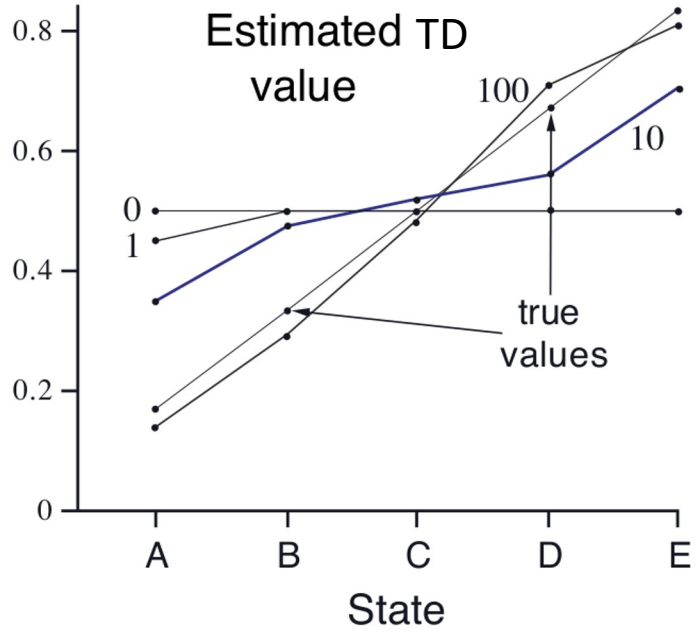
1. express the relationship between the value of a state and its neighbors in the simplest form

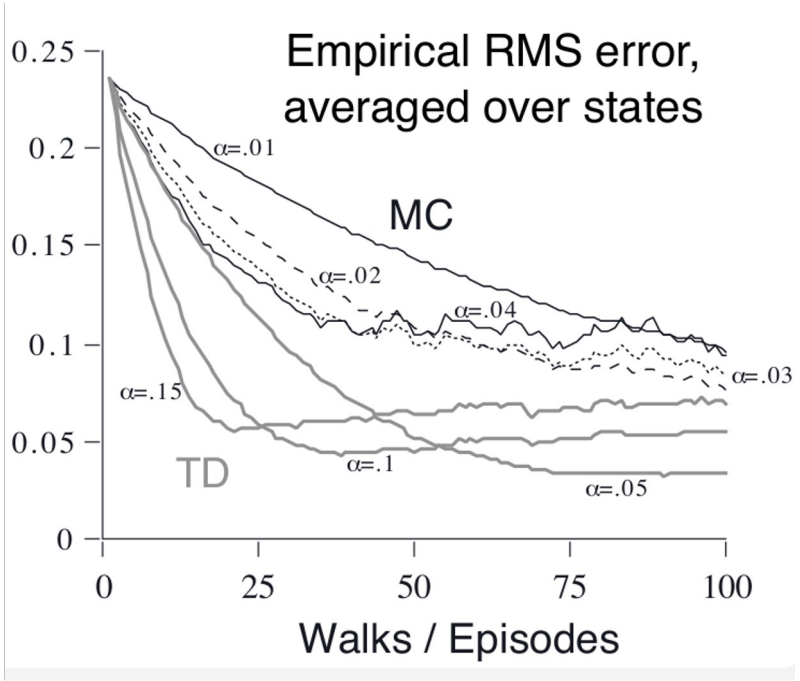
2. say how you could calculate the value of each/all states in closed form

SB Example 6.2: Random Walk

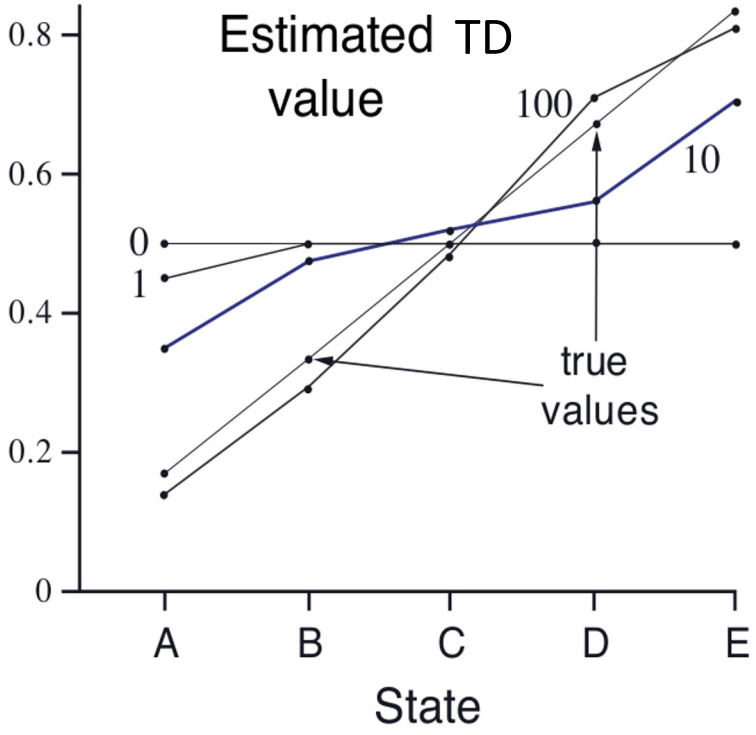


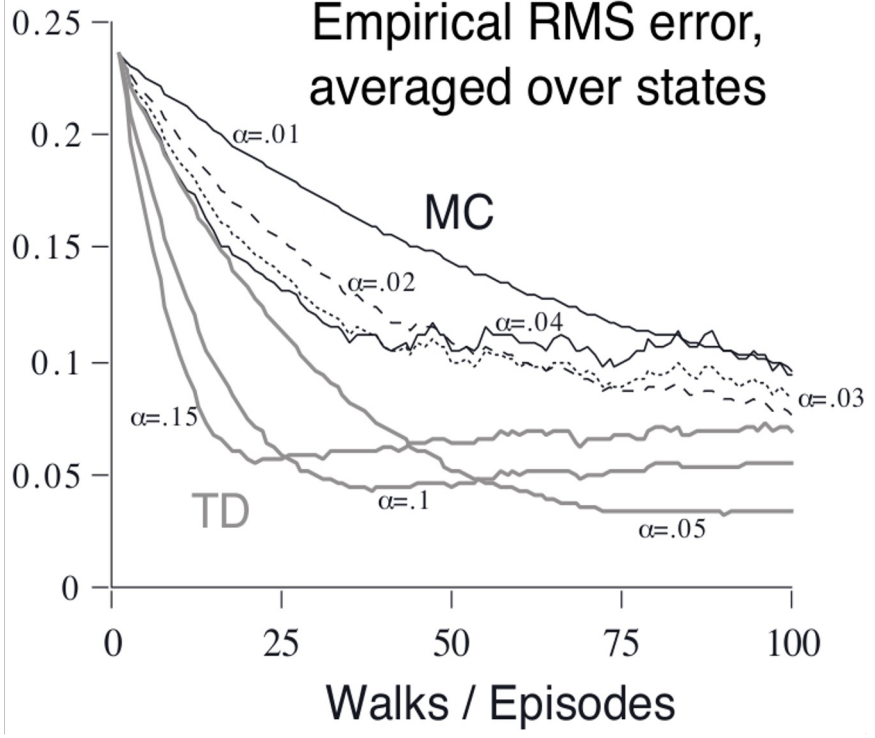
– This is a Markov Reward Process (MDP with no actions)
– Episodes start in state C
– On each time step, there is an equal probability of a left or right transition
– +1 reward at the far right, 0 reward elsewhere
– discount factor of 1
– the true values of the states are: $V(A) = \frac{1}{6}, V(B) = \frac{2}{6}, V(C) = \frac{3}{6}, V(D) = \frac{4}{6}, V(E) = \frac{5}{6},$





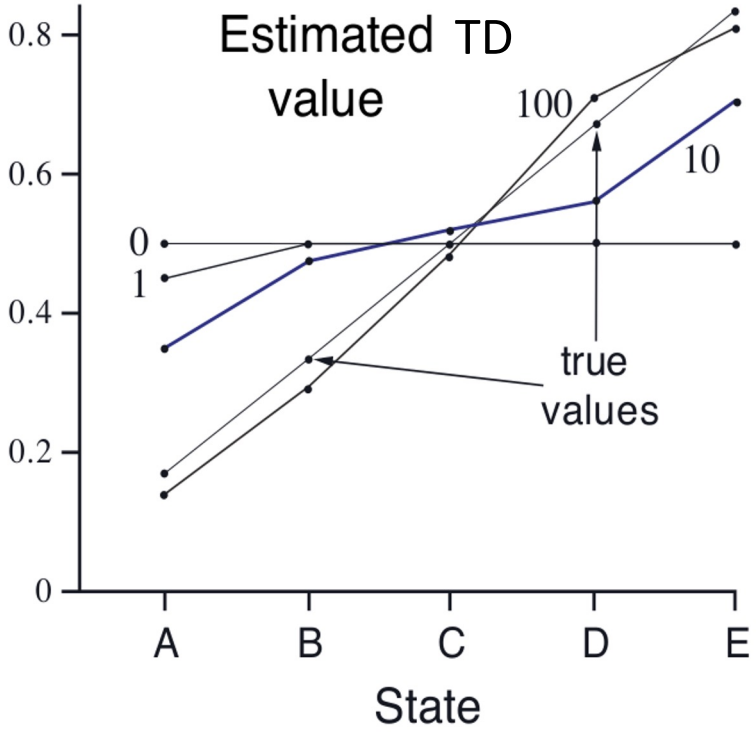
Questions

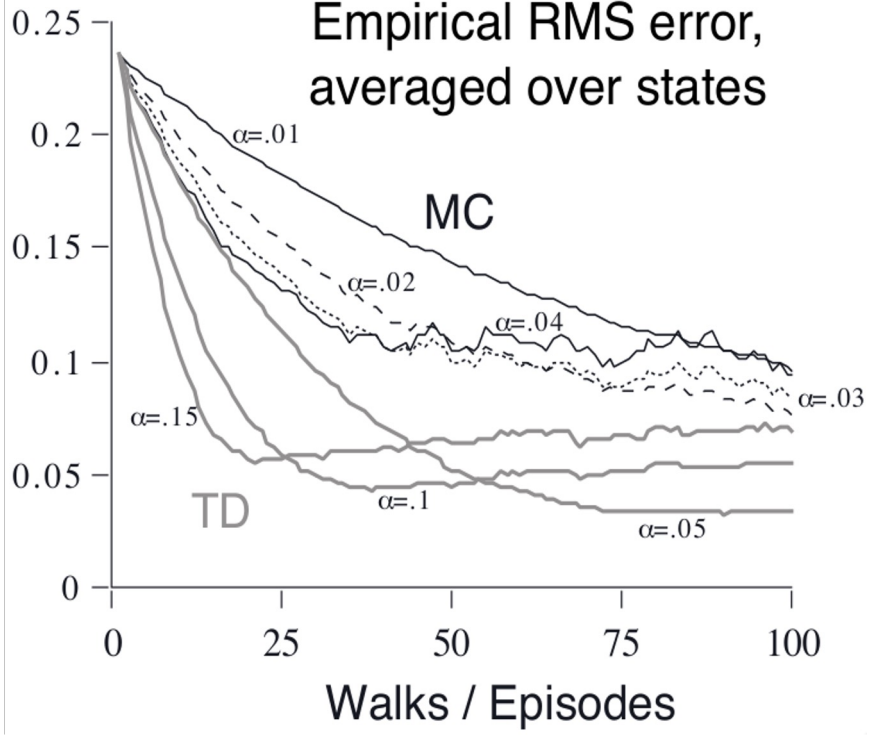




Exercise 6.3 From the results shown in the left graph of the random walk example it appears that the first episode results in a change in only $V(\text{A})$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed? □

Questions

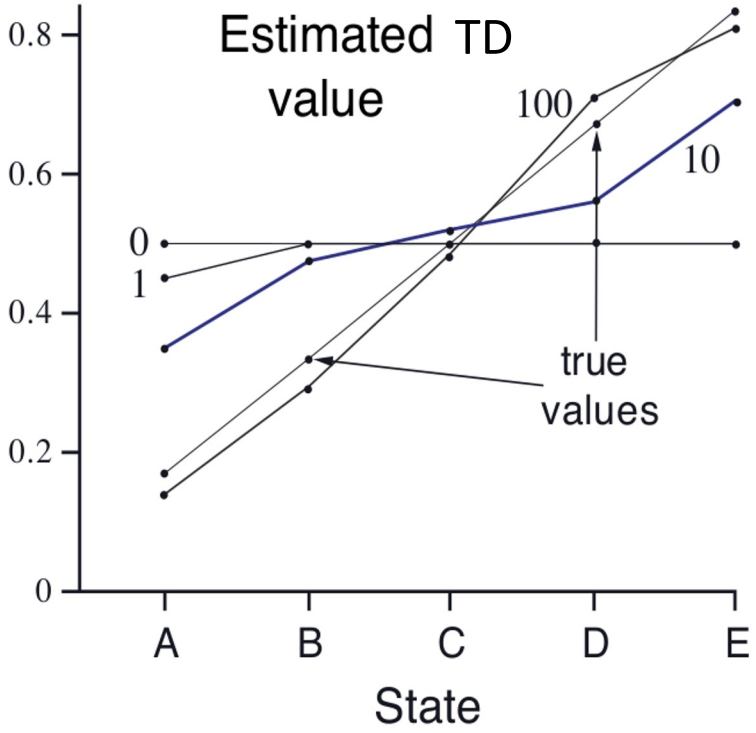


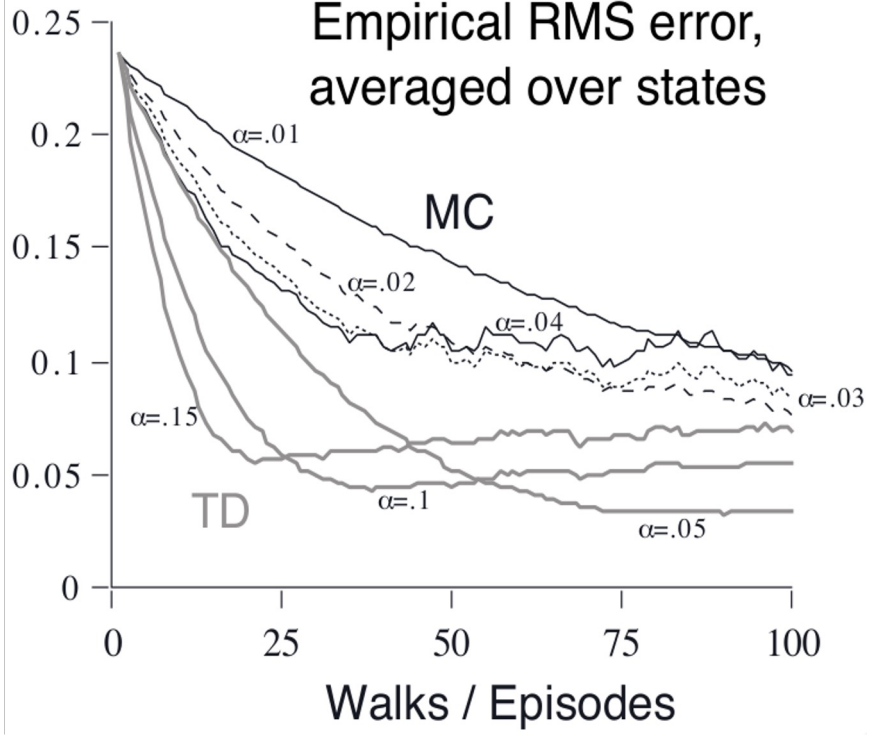


Exercise 6.3 From the results shown in the left graph of the random walk example it appears that the first episode results in a change in only $V(\text{A})$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed? □

α[R + V (terminal) - V (A)] = 0.1[0 + 0 – 0.5] = -.05

Questions





Exercise 6.3 From the results shown in the left graph of the random walk example it appears that the first episode results in a change in only $V(\text{A})$. What does this tell you about what happened on the first episode? Why was only the estimate for this one state changed? By exactly how much was it changed? □

In the figure at right, why do the small-alpha agents converge to lower RMS errors relative to large-alpha agents? Out of the values for alpha shown, which should converge to the lowest RMS value?

The key constraint that MC ignores

Values of adjacent states are closely related:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

Temporal difference learning

How can we use this constraint in a model-free context?

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

Temporal difference learning

How can we use this constraint in a model-free context?

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

Recall that we are trying to estimate:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}^{(i)} \middle| S_t = s \right]$$

$$V_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} \left[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \middle| S_t = s \right]$$

Temporal difference learning

How can we use this constraint in a model-free context?

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

How are these related?

Recall that we are trying to estimate:

$$V_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} \Big[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \,\Big|\, S_t = s \Big]$$

Temporal difference learning

How can we use this constraint in a model-free context?

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

How are these related?

Recall that we are trying to estimate:

$$V_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} \Big[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \,\Big|\, S_t = s \Big]$$

(R_{t+2}, R_{t+3}, …) are sample rewards collected starting in state S_{t+1} while following policy π

$V_\pi(S_{t+1})$ = Expected return starting in state S_{t+1} following policy π

Temporal difference learning

How can we use this constraint in a model-free context?

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}$$

How are these related?

Recall that we are trying to estimate:

$$V_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} \Big[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \Big| S_t = s \Big]$$

(R_{t+2}, R_{t+3}, …) are sample rewards collected starting in state S_{t+1} while following policy π

$V_\pi(S_{t+1})$ = Expected return starting in state S_{t+1} following policy π

$$V_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} \Big[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \Big| S_t = s \Big]$$

Temporal difference learning

Previously: $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \middle| S_t = s\right]$$

Now: $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s\right]$$

Bootstrapping: Update estimate on the basis of other estimates

Temporal difference learning

Previously: $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma R_{t+2}^{(i)} + \gamma^2 R_{t+3}^{(i)} + \ldots \middle| S_t = s\right]$$

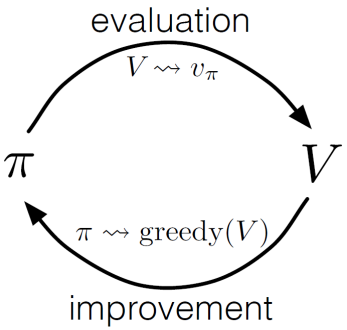
Now: $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s\right]$$

Bootstrapping: Update estimate on the basis of other estimates

Advantages:

- Makes learning more efficient
  (do not have to wait until the end of the episode to learn)
- Enforces local consistency between value function estimates
- Can learn from all experiences in the off-policy case

Our MDP journey



0. Markov decision process $p(s', r \mid s, a)$

1. Dynamic programming
   Value iteration, policy evaluation, policy iteration

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$$

2. Monte-Carlo methods (cf. Multi-armed bandits)
   Sample average of episodic returns

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

3. Temporal-difference learning
   Sample average of <u>one-step returns</u>

$$V_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} \left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s \right]$$

Our MDP journey

3. Temporal-difference learning
   Sample average of one-step returns

$$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)})\middle| S_t = s\right]$$

Typical path from here:

- Batch average → Incremental average
- State-value function → Action-value function
- Prediction → Control
- On-policy → Off-policy

Pros/Cons: TD, MC and DP

DP		MC		TD	
Pro	Con	Pro	Con	Pro	Con
Efficient	Requires full model	Simple	Can be slower than TD	Can be faster than MC	
Complete		Complete	High variance	Complete	
				Low variance	

TD(0) guaranteed to converge to neighborhood of optimal V for a fixed policy if step size parameter is sufficiently small.

– complete just means that it can find an optimal solution

– converges exactly with a step size parameter that decreases in size

Convergence/correctness of TD(0)

It will be easier to have this discussion if I introduce a batch version of TD(0)…

Online TD(0)

TD(0) for estimating V^π:

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

This algorithm runs online. It performs one TD update per experience

Batch TD(0)

$\mathbb{D}$ is a dataset of experience

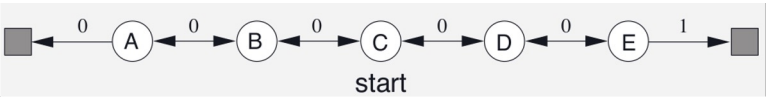
Batch updating:
Collect a dataset $\mathbb{D}$ of experience (somehow)
Initialize *V* arbitrarily
Repeat until *V* converged:
    ∂=0
    For all $(s, a, s', r) \in \mathbb{D}$:

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

$$V = V + \partial$$

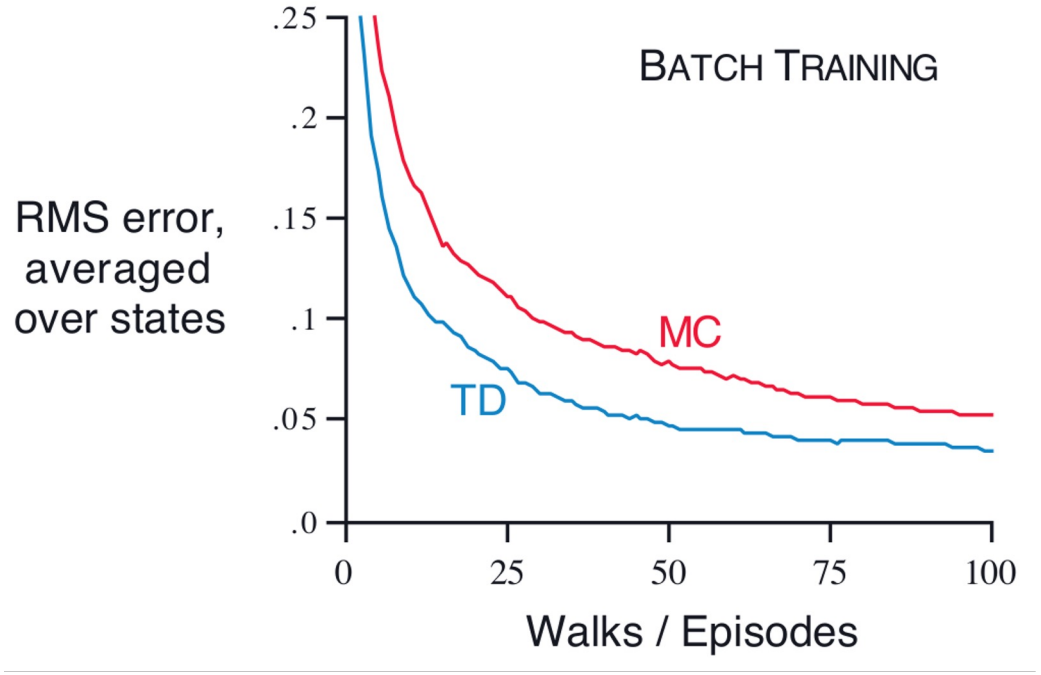
This integrates a bunch of TD steps into one update

Let's consider the case where we have a fixed dataset of experience
    – all our learning must leverage a fixed set of experiences
    – this is similar to our two-array DP methods, but update after each batch and repeat on the same data

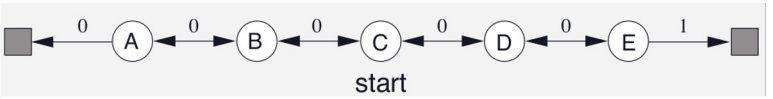


TD(0)/MC comparison

Batch TD(0) and batch MC both converge for sufficiently small step size
– but they converge to different answers!

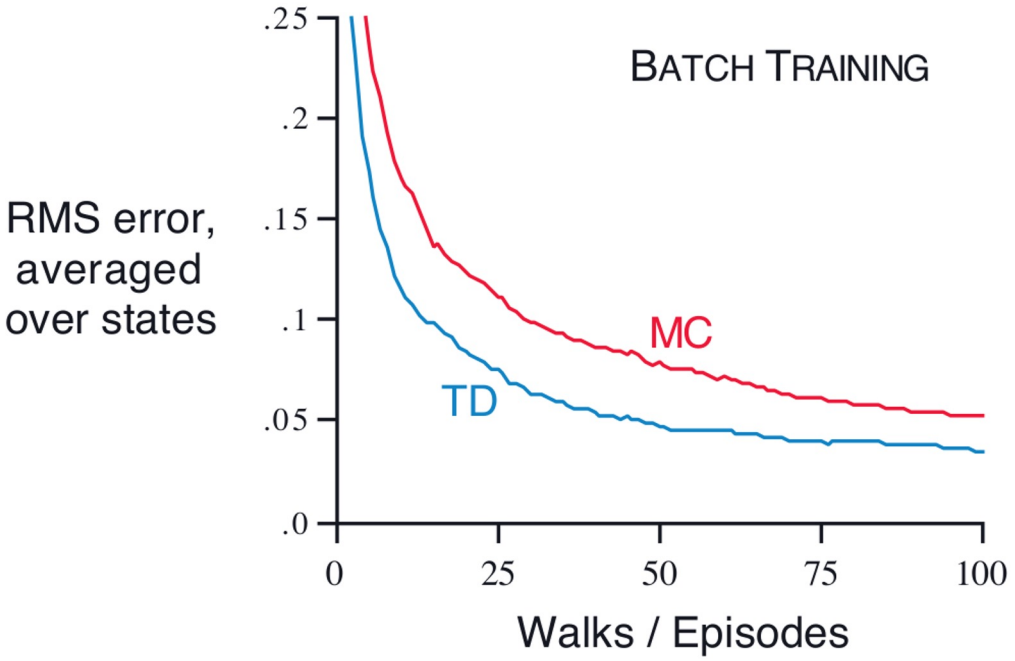


After each new episode, all previous episodes were treated as a batch, and algorithm was trained until convergence. All repeated 100 times.



Question

Batch TD(0) and batch MC both converge for sufficiently small step size
– but they converge to different answers!



After each new episode, all previous episodes were treated as a batch, and algorithm was trained until convergence. All repeated 100 times.

Why?

Think-pair-share

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

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

Calculate:

1. batch first-visit MC estimates for V(A) and V(B)

2. batch TD(0) estimates for V(A) and V(B)

Are these the same? Different? Why?

Different estimates

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

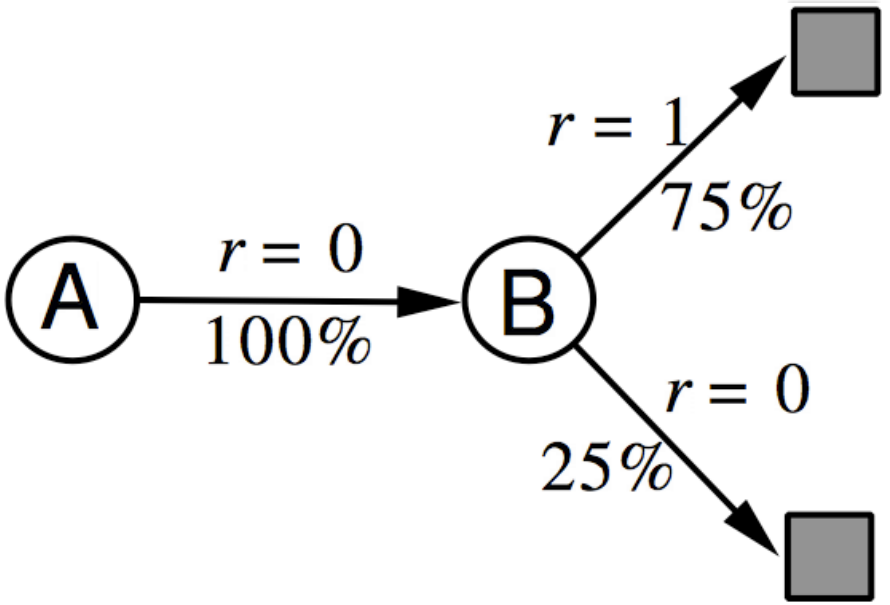
$V(\mathrm{B})?$ 0.75

$V(\mathrm{A})?$ 0?

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

TD estimate

- Modeling the Markov process based on the observed training data



$V(\mathrm{A})?$ 0.75

Optimality of batch TD(0)

- The prediction that best matches the training data is V(A)=0:
  - This minimizes the **mean-square-error** between V(s) and the sample returns in the training set. (**zero** MSE in our example)
  - Under batch training, this is what constant-α MC gets
- TD(0) achieves a different type of optimality, where V(A)=0.75
  - This is correct for the maximum likelihood estimate of the Markov model generating the data
  - i.e., if we do a best fit Markov model, and assume it is exactly correct, and then compute the predictions
  - This is called the **certainty-equivalence estimate**
  - This is what TD gets

Optimality of batch TD(0)

- The prediction that best matches the training data is V(A)=0:
  - This minimizes the **mean-square-error** between V(s) and the sample returns in the training set. (**zero** MSE in our example)
  - Under batch training, this is what constant-α MC gets
- TD(0) achieves a different type of optimality, where V(A)=0.75
  - This is correct for the maximum likelihood estimate of the Markov model generating the data
  - i.e., if we do a best fit Markov model, and assume it is exactly correct, and then compute the predictions
  - This is called the **certainty-equivalence estimate**
  - This is what TD gets

This analysis doesn't quite hold in the online case…

Back to the online TD case…

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Batch average→ Incremental average

Suppose we have a random variable X
and we want to estimate the mean from samples $x_1, \ldots, x_k$

After k samples $$\hat{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i$$

Can show that $$\hat{x}_k = \hat{x}_{k-1} + \frac{1}{k}(x_k - \hat{x}_{k-1})$$

Can be written as $$\hat{x}_k = \hat{x}_{k-1} + \alpha(k)(x_k - \hat{x}_{k-1})$$

Update rule $$\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x})$$

Applied to $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s\right]$$

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

Recall that a sample consists of (s, a, s', r)

Batch average→ Incremental average

Suppose we have a random variable X
and we want to estimate the mean from samples $x_1, \ldots, x_k$

After k samples $\hat{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i$

Can show that $\hat{x}_k = \hat{x}_{k-1} + \frac{1}{k}(x_k - \hat{x}_{k-1})$

Can be written as $\hat{x}_k = \hat{x}_{k-1} + \alpha(k)(x_k - \hat{x}_{k-1})$

Update rule $\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x})$

Bootstrap estimate
(1-step return)

Applied to $V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s\right]$

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

Recall that a sample consists of (s, a, s', r)

Batch average→ Incremental average

Suppose we have a random variable X
and we want to estimate the mean from samples $x_1, \ldots, x_k$

After k samples $$\hat{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i$$

Can show that $$\hat{x}_k = \hat{x}_{k-1} + \frac{1}{k}(x_k - \hat{x}_{k-1})$$

Can be written as $$\hat{x}_k = \hat{x}_{k-1} + \alpha(k)(x_k - \hat{x}_{k-1})$$

Update rule $$\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x})$$

Current average
(V = Avg. return)

Applied to $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s\right]$$

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

Recall that a sample consists of (s, a, s', r)

Batch average→ Incremental average

Suppose we have a random variable X
and we want to estimate the mean from samples $x_1, \ldots, x_k$

After k samples $$\hat{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i$$

Can show that $$\hat{x}_k = \hat{x}_{k-1} + \frac{1}{k}(x_k - \hat{x}_{k-1})$$

Can be written as $$\hat{x}_k = \hat{x}_{k-1} + \alpha(k)(x_k - \hat{x}_{k-1})$$

Update rule $$\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x})$$

New average
(V = Avg. return)

Applied to $$V_\pi(s) = \frac{1}{N}\sum_{i=1}^{N}\left[R_{t+1}^{(i)} + \gamma V_\pi(S_{t+1}^{(i)}) \middle| S_t = s\right]$$

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

Recall that a sample consists of (s, a, s', r)

(Online) TD learning

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

Value prediction based on neighbor's information

Our own current prediction

Temporal-difference (TD) error

Our MDP journey

3. Temporal-difference learning
   Moving average of one-step returns

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

Typical path from here:

✓ Batch average → Incremental average

- State-value function → Action-value function
- Prediction → Control
- On-policy → Off-policy

V → Q ; Prediction → Control

Temporal-difference (TD) RL:

V = Moving average of one-step returns:

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

We finally have a good estimate of the value function!

How to pick actions?

V → Q ; Prediction → Control

Temporal-difference (TD) RL:

V = Moving average of one-step returns:

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

We finally have a good estimate of the value function!

How to pick actions? The method from value iteration:

$$\text{Output a deterministic policy, } \pi \approx \pi_*, \text{ such that}$$
$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$$

V → Q ; Prediction → Control

Temporal-difference (TD) RL:

V = Moving average of one-step returns:

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

We finally have a good estimate of the value function!

How to pick actions? The method from value iteration:

$$\text{Output a deterministic policy, } \pi \approx \pi_*, \text{ such that}$$
$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$$

We do not have a model!

V → Q ; Prediction → Control

Temporal-difference (TD) RL:

V = Moving average of one-step returns:

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

We finally have a good estimate of the value function!

How to pick actions? The method from value iteration:

$$\text{Output a deterministic policy, } \pi \approx \pi_*, \text{ such that}$$
$$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$$

We do not have a model!

Recall Q-functions:

$$a^* = \arg\max_{a \in A} Q(s, a)$$

V → Q ; Prediction → Control

Temporal-difference (TD) RL:

V = Moving average of one-step returns:

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

We finally have a good estimate of the value function!

How to pick actions?

Recall Q-functions: $$a^* = \arg\max_{a \in A} Q(s, a)$$

What is the analog of TD for Q-functions?

V → Q ; Prediction → Control

Temporal-difference (TD) RL:

V = Moving average of one-step returns:

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

We finally have a good estimate of the value function!

How to pick actions?

Recall Q-functions:

$$a^* = \arg\max_{a \in A} Q(s, a)$$

What is the analog of TD for Q-functions?

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

SARSA: TD Learning for Control

Recall the two types of value function:

1) state-value function: $V^{\pi}(s)$

2) action-value function: $Q^{\pi}(s, a)$

State-value fn

Action-value fn

SARSA: TD Learning for Control

Recall the two types of value function:

1) state-value function: $V^\pi(s)$ State-value fn

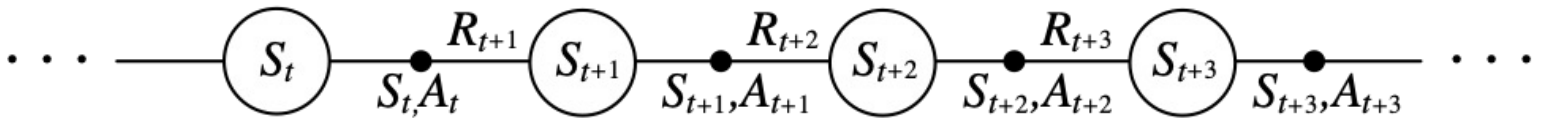
2) action-value function: $Q^\pi(s,a)$ Action-value fn

Update rule for TD(0):

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

Update rule for SARSA:

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



SARSA: TD Learning for Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

SARSA: TD Learning for Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

How do I choose actions? Why does this converge to q_*?

SARSA: TD Learning for Control

SARSA:

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Convergence: guaranteed to converge for any e-soft policy (such as ε-greedy) w/ $\varepsilon > 0$

– strictly speaking, we require the probability of visiting any state-action pair to be greater than zero always

– also require the policy to converge to the greedy policy (e.g., ε goes to 0)

SARSA: TD Learning for Control

SARSA:

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each
    Take action A
    Choose A' fro
    $Q(S,A) \leftarrow Q($
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

ε-soft policy: any policy for which $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)}$

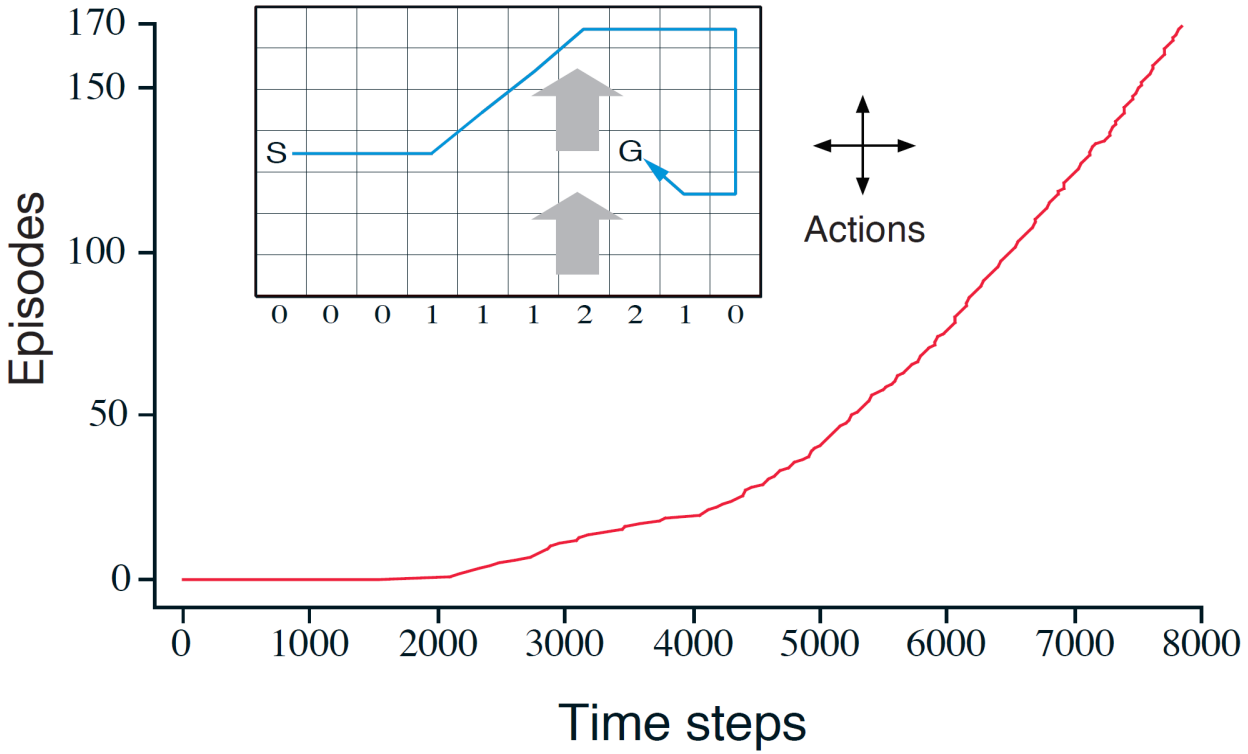
Convergence: guaranteed to converge for any e-soft policy (such as ε-greedy) w/ $\varepsilon > 0$

– strictly speaking, we require the probability of visiting any state-action pair to be greater than zero always

– also require the policy to converge to the greedy policy (e.g., ε goes to 0)

SARSA Example: Windy Gridworld

Shows # of episodes completed by the given number of time steps



– reward = -1 for all transitions until termination at goal state
– undiscounted, deterministic transitions
– episodes only terminate at goal state
– use a ε -greedy policy with ε=0.1

– this would be hard to solve using MC b/c episodes are very long or never complete
– optimal path length from start to goal: 15 time steps
– average path length 17 time steps (why is this longer?)

Our MDP journey

3. Temporal-difference learning
   Moving average of one-step returns

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

Typical path from here:

- ✓ Batch average → Incremental average
- ✓ State-value function → Action-value function
- ✓ Prediction → Control
- - On-policy → Off-policy

How to pick the best actions?
Q-learning

On-policy → Off-policy

The following equation led to TD-learning:

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

What is the analog of TD for Q-functions?

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

This only gives Q-values for policy π
but we want Q* (for policy π*)

On-policy → Off-policy

Recall: $V^{\pi}(s)$ = Expected return starting in state s and following policy π thereafter

$Q^{\pi}(s,a)$ = Expected return starting in s, taking action a, and following policy π thereafter

Properties: $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$

On-policy → Off-policy

Recall: $V^{\pi}(s)$ = Expected return starting in state s and following policy π thereafter

$Q^{\pi}(s,a)$ = Expected return starting in s, taking action a, and following policy π thereafter

Properties:

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

$$V^{\pi^*}(s) = Q^{\pi^*}(s, \pi^*(s)) = \max_{a \in A} Q^{\pi^*}(s,a)$$

On-policy → Off-policy

Recall: $V^{\pi}(s)$ = Expected return starting in state s and following policy π thereafter

$Q^{\pi}(s,a)$ = Expected return starting in s, taking action a, and following policy π thereafter

Properties:

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

$$V^{\pi^*}(s) = Q^{\pi^*}(s, \pi^*(s)) = \max_{a \in A} Q^{\pi^*}(s,a)$$

Previously, our bootstrap estimate is: $r + \gamma \hat{V}(s')$

Now, change it to: $r + \gamma \max_{a' \in A} \hat{Q}(s', a')$

On-policy → Off-policy

The following equation led to TD-learning:

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

What is the analog of TD for Q-functions?

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

This only gives Q-values for policy π
but we want Q* (for policy π*)

To find the optimal Q-function Q* and optimal policy π*:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

Q-Learning (Watkins, 1989)

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

Q-Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

Unlike SARSA, we do not need a' any more,
since we maximize over next actions a'

Q-Learning: A Variation on SARSA

Update rule for TD(0):

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

Update rule for SARSA:

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

Update rule for Q-Learning:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \boxed{\max_{a} Q^{\pi}(s_{t+1}, a)} - Q^{\pi}(s_t, a_t)]$$

This is the only difference between SARSA and Q-Learning

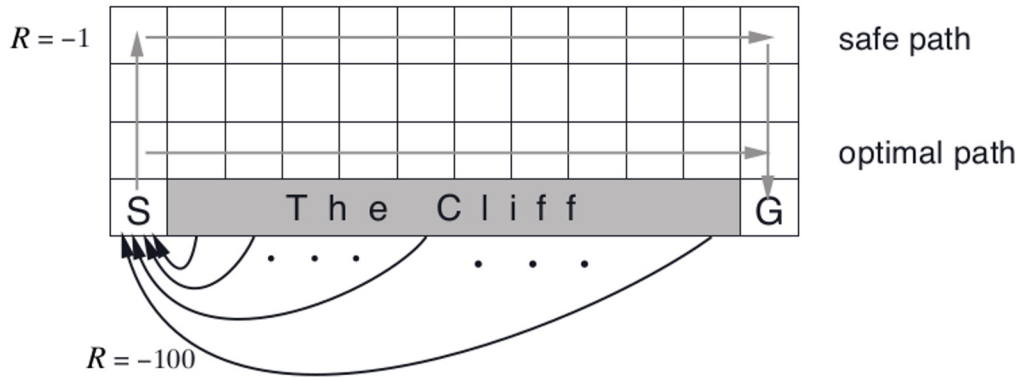
Q-Learning Convergence

- Q-learning is the off-policy version of SARSA
- Converges to q* without needing to adjust behavior policy
- Convergence just requires positive probability for visiting all state-action pairs (i.e., all Q-values continue to be updated)

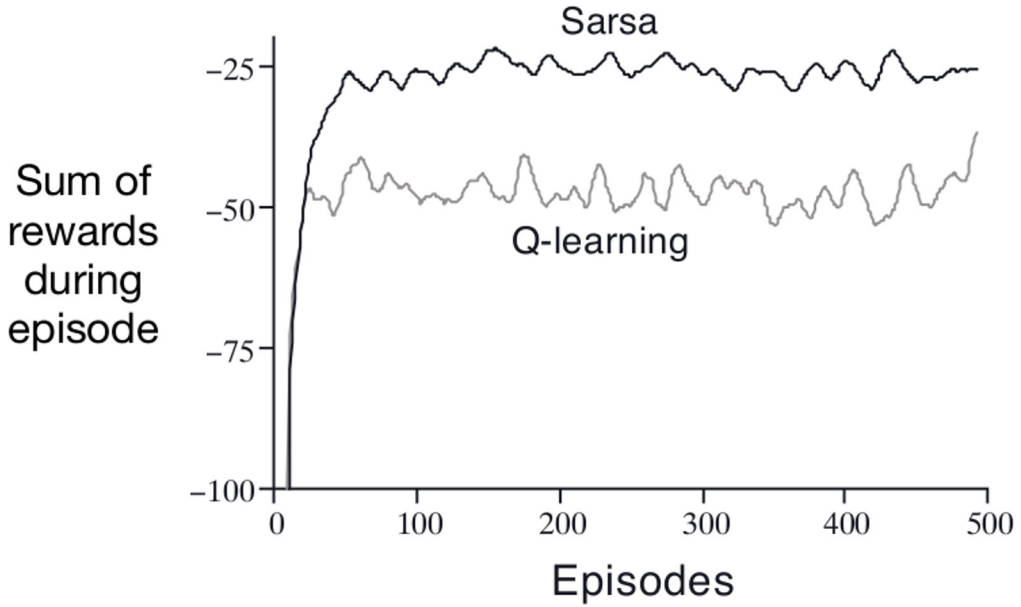
Q-Learning example

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Think-pair-share: Cliffworld



– deterministic actions
– -1 reward per time step;
-100 reward for falling off cliff
– e-greedy action selection
(with e=0.1)



Why does Q-learning get less avg reward?

What policies do Q-learning and Sarsa learn?

In what sense are each of these solutions optimal?

How would these results be different for different values of epsilon?

Learning targets

Monte-Carlo:

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

TD(0):

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

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Learning targets

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

Monte-Carlo:

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

TD(0):

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

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

$$q_\pi(s, a) \quad = \quad \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

Learning targets

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

Monte-Carlo:

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

TD(0):

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

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

$$q_\pi(s, a) \quad = \quad \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

Is there anything else with the same expected value?

Expected SARSA

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

Expected SARSA

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

$$q_\pi(s, a) \quad = \quad \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

Expected SARSA

$$q_\pi(s,a) \quad = \quad \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

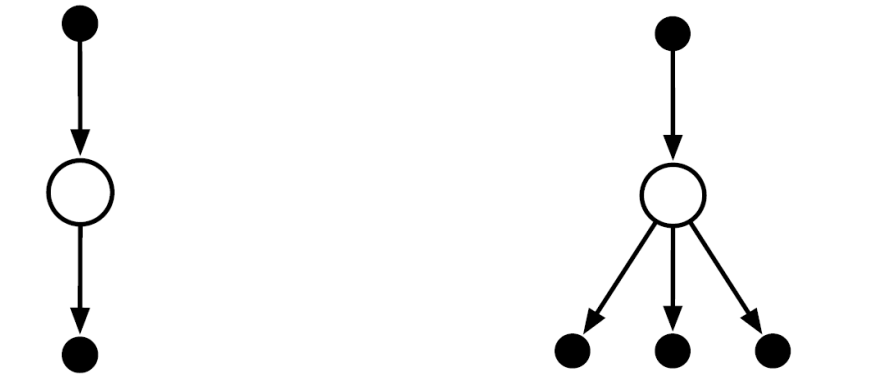
SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big]$$

$$\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big],$$



Sarsa Expected Sarsa

Expected SARSA

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

SARSA samples the next action

Expected SARSA considers all possible next actions

Expected SARSA

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

SARSA samples the next action
Expected SARSA considers all possible next actions

"Expected SARSA moves deterministically
in the same direction as SARSA moves in expectation."

Expected SARSA

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

"Expected SARSA moves deterministically
in the same direction as SARSA moves in expectation."

More computation, lower variance (no randomness in A_{t+1})

Expected SARSA

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

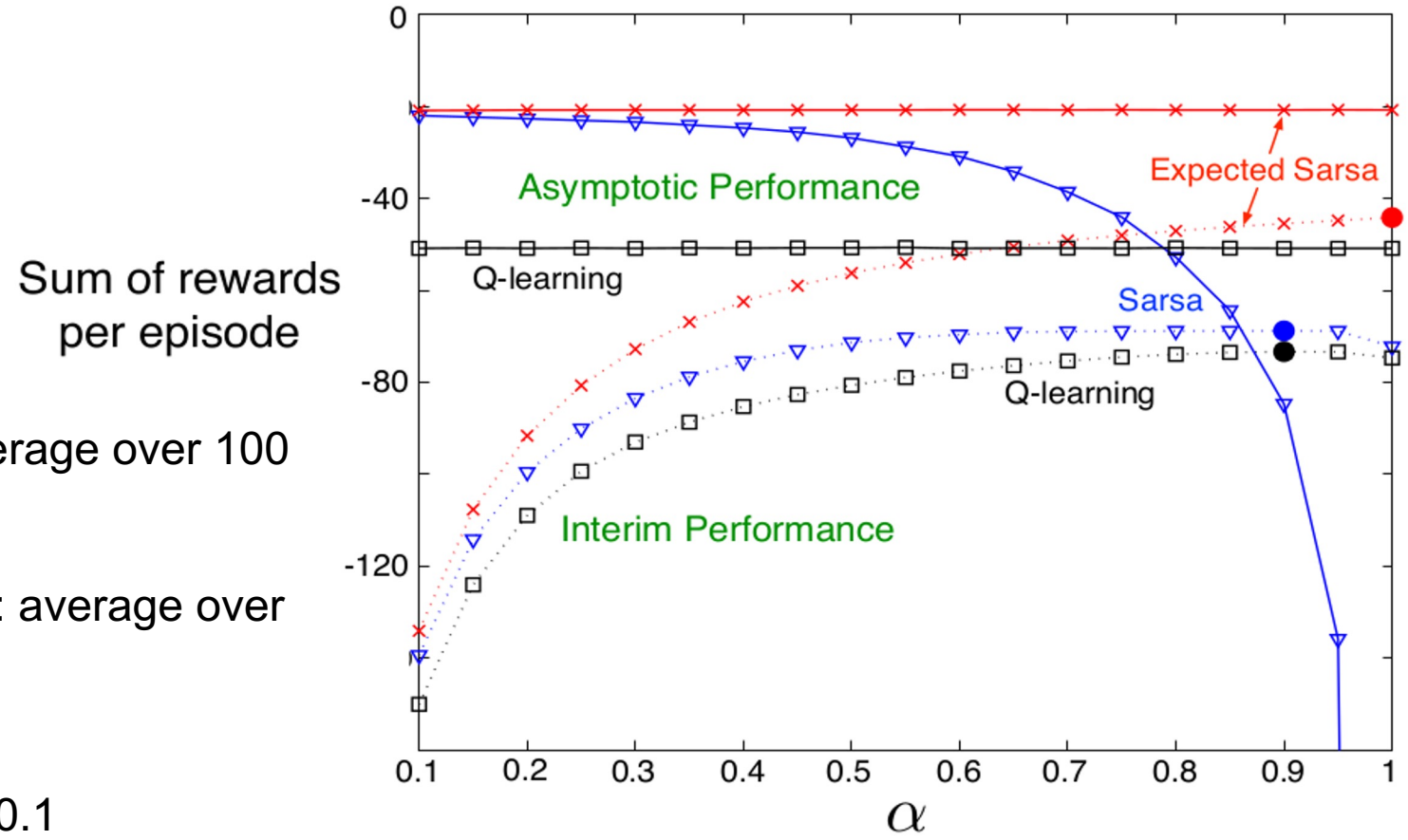
"Expected SARSA moves deterministically
in the same direction as SARSA moves in expectation."

More computation, lower variance (no randomness in A_{t+1})

Note: Expected SARSA can be used on or off-policy

Expected SARSA

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)]$$



Interim performance: average over 100 episodes

Asymptotic performance: average over 100k episodes

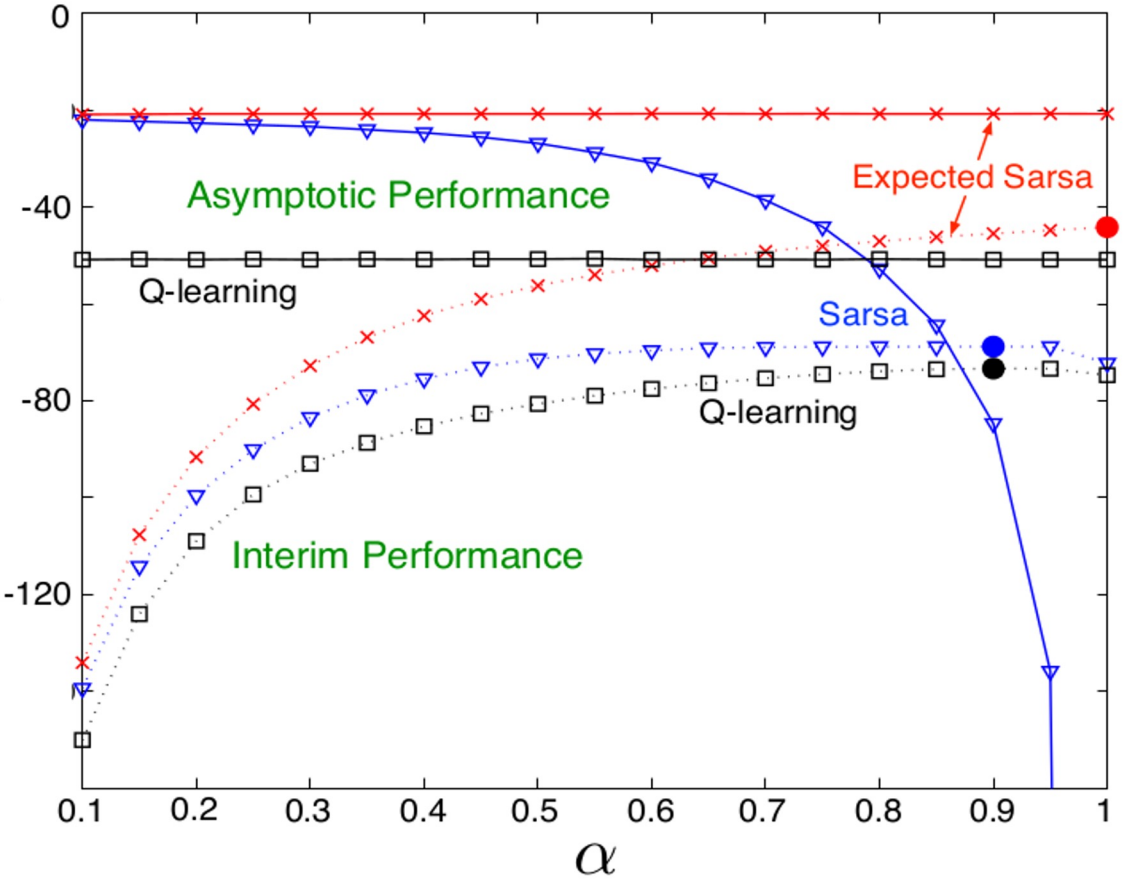
Details:
– cliff walking task, eps=0.1

Expected SARSA

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)]$$

Transitions are deterministic

Sum of rewards per episode

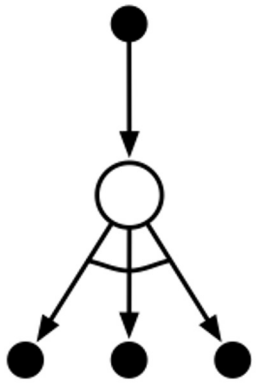


Interim performance: average over 100 episodes

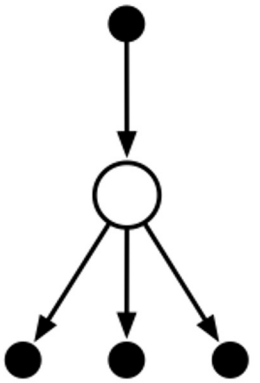
Asymptotic performance: average over 100k episodes

Details:
– cliff walking task, eps=0.1

Backup diagrams



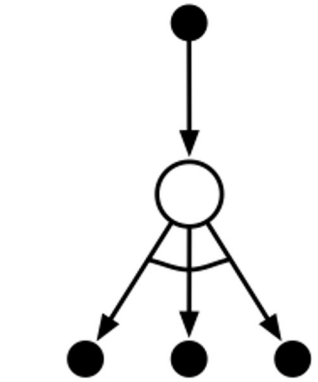
Q-learning



Expected Sarsa

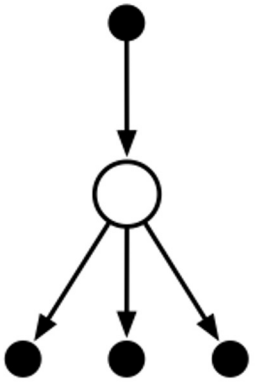
Backup diagrams

Max



Q-learning

Expectation



Expected Sarsa

Maximization bias in Q-learning

The problem:

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

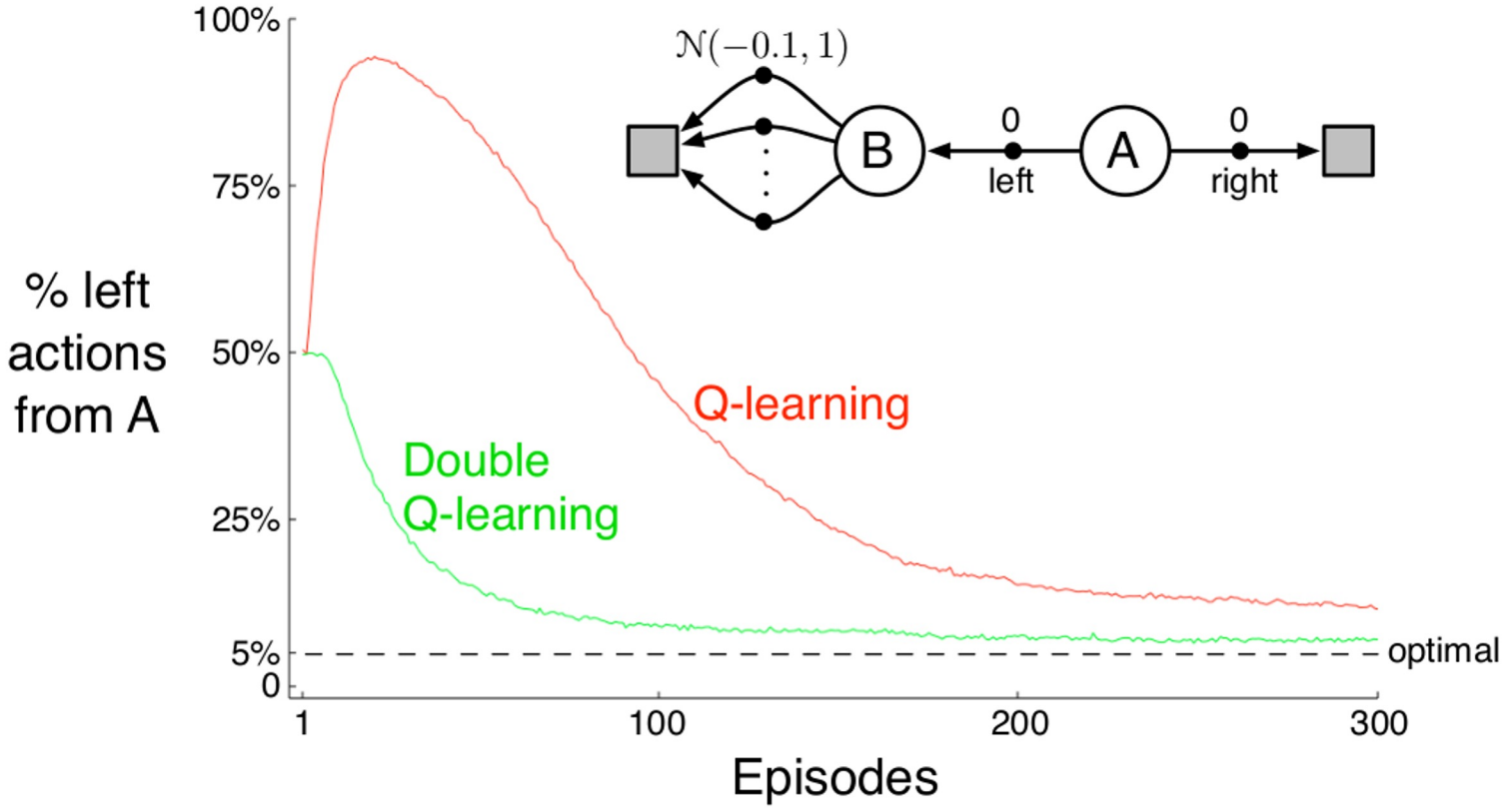
Maximization bias in Q-learning

Maximization over random samples
is not a good estimate of the max
of the expected values

The problem:

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

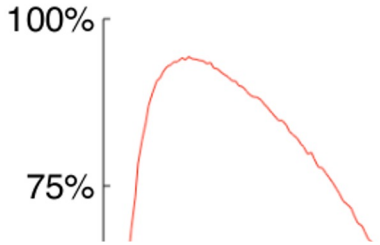
Why maximization bias is a problem

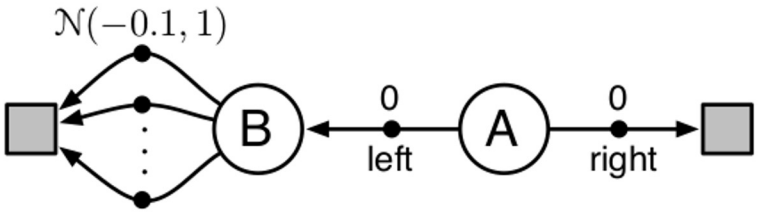


$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)]$$

Two states, two actions
Start at A
Rewards:
– going right from A always leads to zero reward and then terminates
– lots of left actions from B. All have stochastic reward with mean=-0.1 and unit variance and then terminates

Why maximization bias is a problem





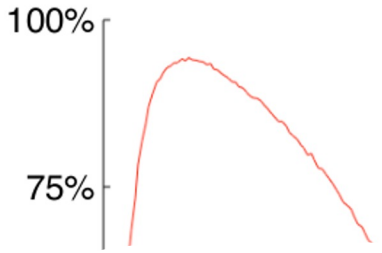
What are the true values of the states?

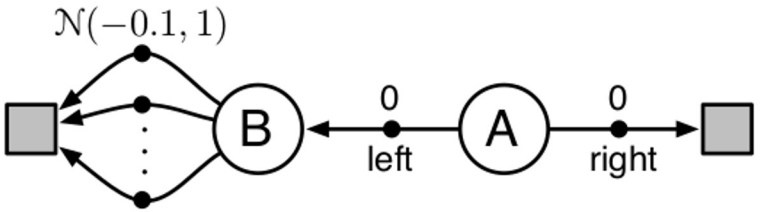
What if there were more left actions from B?

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

Two states, two actions
Rewards:
– going right from A always leads to zero reward and then terminates
– lots of left actions from B. All have stochastic reward with mean=-0.1 and unit variance and then terminates

Why maximization bias is a problem





% left actions from A

How do we fix this problem?

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

Two states, two actions
Rewards:
– going right from A always leads to zero reward and then terminates
– lots of left actions from B. All have stochastic reward with mean=-0.1 and unit variance and then terminates

Double Q-Learning

Double Q-Learning:

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

    else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Question

Double Q-Learning:

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

    else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

How exactly does this fix the problem?

Question

Double Q-Learning:

Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

    else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Not using the same estimator for max and expectation

Maximization bias in Q-learning

Maximization over random samples
is not a good estimate of the max
of the expected values

The problem:

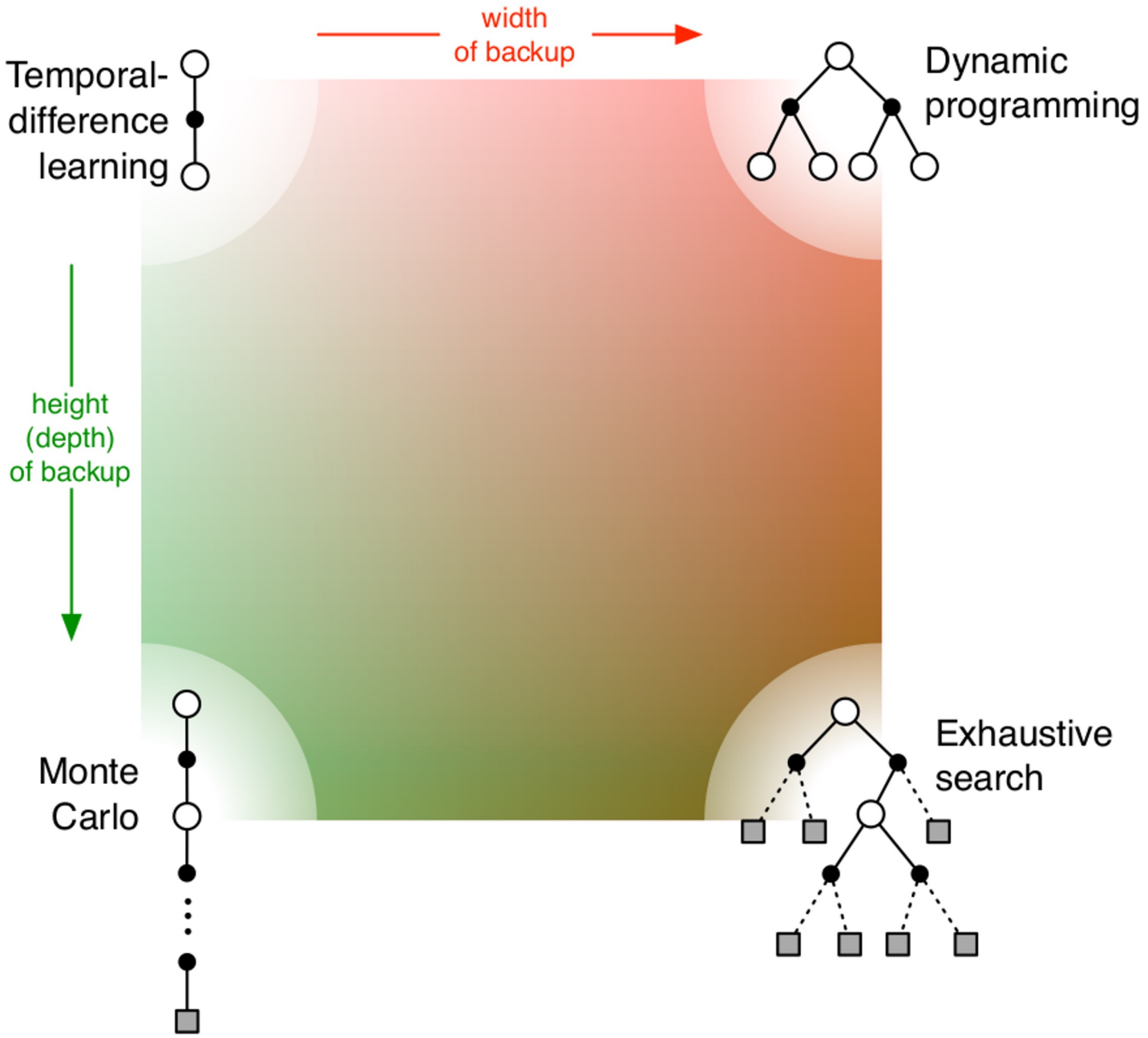
$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)]$$

For example: suppose you have two Gaussian variables, *a* and *b*.

Suppose that you want to estimate the expected value of the max over the two variables
- but you only get samples from each of the variables. You don't know the expectation for either variable

Solution:
- estimate sample mean of each variable, $\hat{a}$ and $\hat{b}$
- then, calculate $\max(\hat{a}, \hat{b})$

Unified view



Intermediate summary

- Introduced one-step tabular model-free TD methods
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are computationally efficient
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data
- Extend prediction to control by employing some form of GPI
    - On-policy control: **Sarsa**, **Expected Sarsa**
    - Off-policy control: **Q-learning**, **Expected Sarsa**
- Avoiding maximization bias with **Double Q-learning**

Learning targets

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

Monte-Carlo:

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

TD(0):

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

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

Q-learning (off-policy):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

$$q_*(s, a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big]$$

Are there more learning targets?

Monte-Carlo:

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

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

TD(0):

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

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

$$q_\pi(s,a) \quad = \quad \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

Q-learning (off-policy):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

$$q_*(s,a) \quad = \quad \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big]$$

TD vs MC

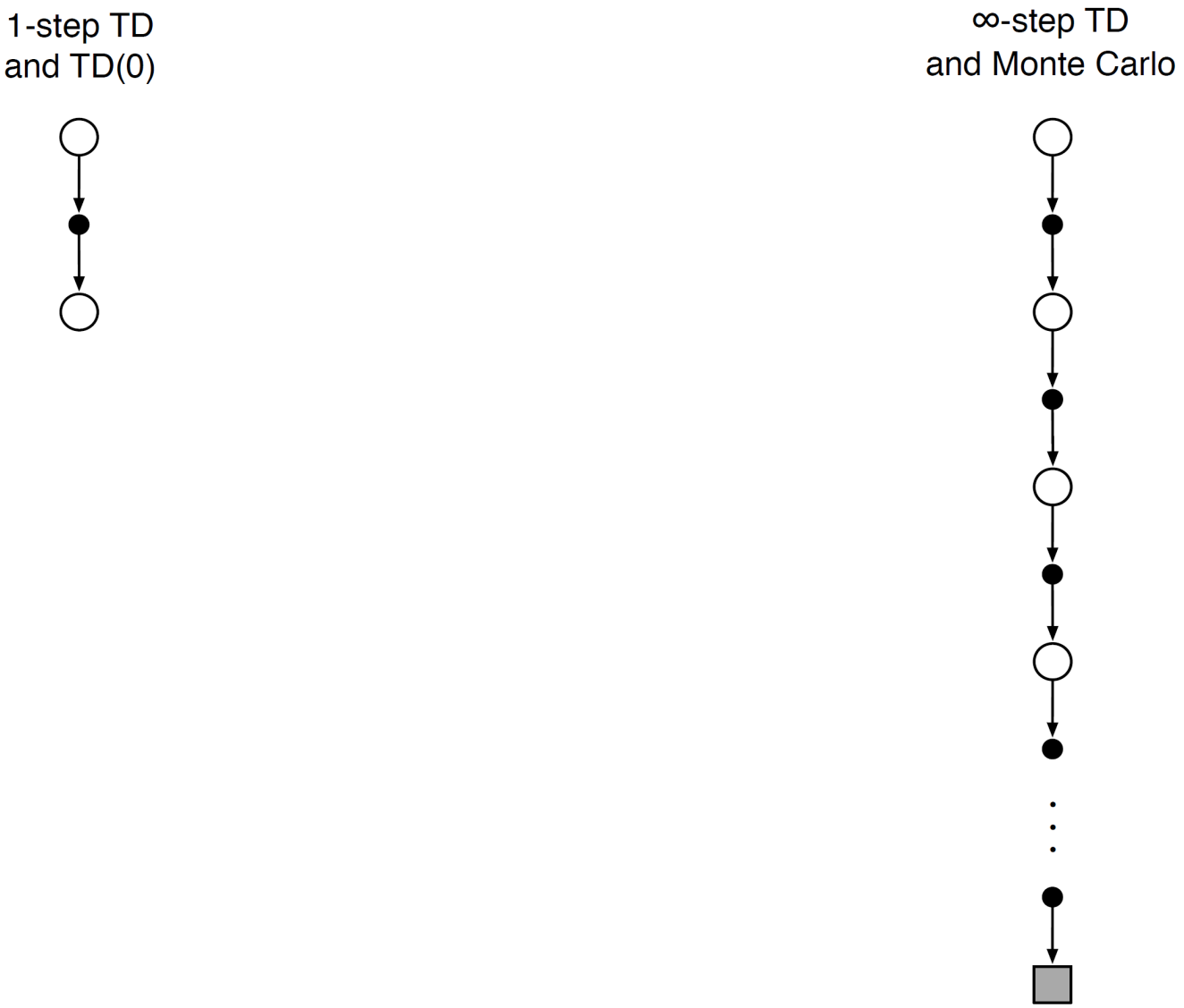


Figure 7.1: The backup diagrams of n-step methods. These methods form a spectrum ranging from one-step TD methods to Monte Carlo methods.

n-step returns

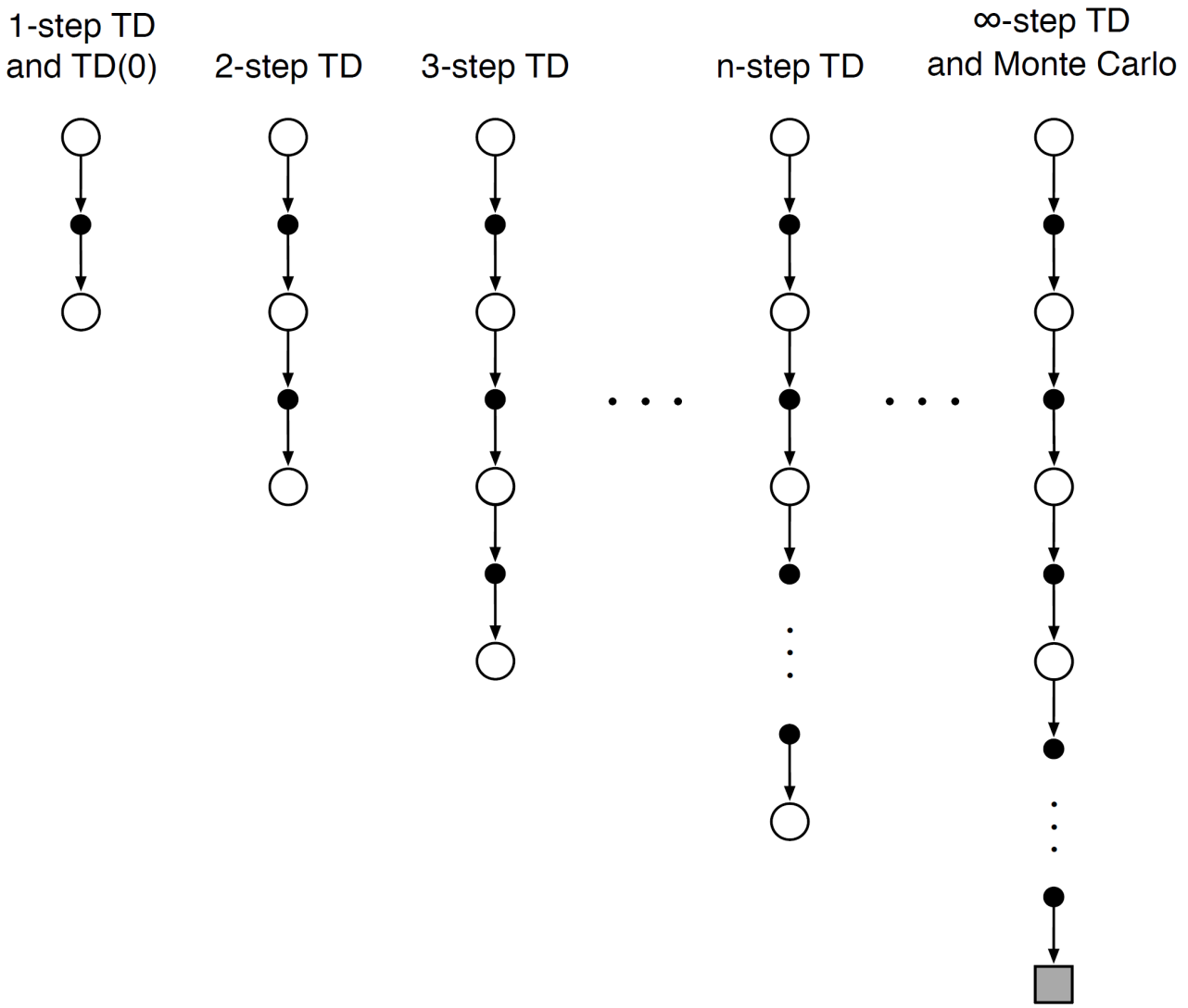


Figure 7.1: The backup diagrams of n-step methods. These methods form a spectrum ranging from one-step TD methods to Monte Carlo methods.

n-step returns

Monte-Carlo return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

1-step return:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

n-step returns

Monte-Carlo return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

1-step return:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

2-step return:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

n-step returns

Monte-Carlo return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

1-step return:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

2-step return:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

...

n-step return:

$$G_{t:t+n} \doteq G_t \text{ if } t+n \geq T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

n-step returns

Similarly, the target for an arbitrary n-step update is the *n-step return*:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \tag{7.1}$$

for all n, t such that $n \geq 1$ and $0 \leq t < T - n$. All n-step returns can be considered approximations to the full return, truncated after n steps and then corrected for the remaining missing terms by $V_{t+n-1}(S_{t+n})$. If $t + n \geq T$ (if the n-step return extends to or beyond termination), then all the missing terms are taken as zero, and the n-step return defined to be equal to the ordinary full return ($G_{t:t+n} \doteq G_t$ if $t + n \geq T$).

n-step returns

Similarly, the target for an arbitrary n-step update is the *n-step return*:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \tag{7.1}$$

for all n, t such that $n \geq 1$ and $0 \leq t < T - n$. All n-step returns can be considered approximations to the full return, truncated after n steps and then corrected for the remaining missing terms by $V_{t+n-1}(S_{t+n})$. If $t + n \geq T$ (if the n-step return extends to or beyond termination), then all the missing terms are taken as zero, and the n-step return defined to be equal to the ordinary full return ($G_{t:t+n} \doteq G_t$ if $t + n \geq T$).

Note that n-step returns for $n > 1$ involve future rewards and states that are not available at the time of transition from t to $t + 1$. No real algorithm can use the n-step return until after it has seen R_{t+n} and computed V_{t+n-1}. The first time these are available is $t + n$.

n-step returns

Similarly, the target for an arbitrary n-step update is the *n-step return*:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \tag{7.1}$$

for all n, t such that $n \geq 1$ and $0 \leq t < T - n$. All n-step returns can be considered approximations to the full return, truncated after n steps and then corrected for the remaining missing terms by $V_{t+n-1}(S_{t+n})$. If $t + n \geq T$ (if the n-step return extends to or beyond termination), then all the missing terms are taken as zero, and the n-step return defined to be equal to the ordinary full return ($G_{t:t+n} \doteq G_t$ if $t + n \geq T$).

Note that n-step returns for $n > 1$ involve future rewards and states that are not available at the time of transition from t to $t + 1$. No real algorithm can use the n-step return until after it has seen R_{t+n} and computed V_{t+n-1}. The first time these are available is $t + n$. The natural state-value learning algorithm for using n-step returns is thus

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \leq t < T, \tag{7.2}$$

while the values of all other states remain unchanged: $V_{t+n}(s) = V_{t+n-1}(s)$, for all $s \neq S_t$.

Learning targets

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

Monte-Carlo:

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

TD(0):

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

n-step returns:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \le t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$G_{t:t+n} \doteq G_t \text{ if } t + n \ge T$$

n-step TD (prediction)

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | | Take an action according to $\pi(\cdot|S_t)$
    | | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | | If S_{t+1} is terminal, then $T \leftarrow t + 1$

n-step TD (prediction)

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$

Example: Grid world

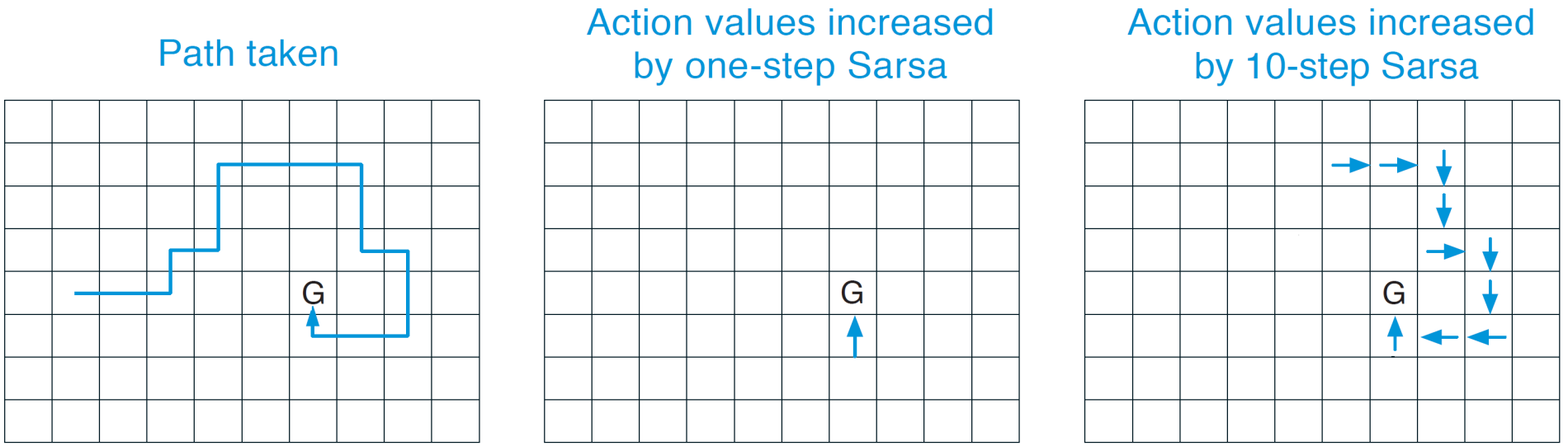
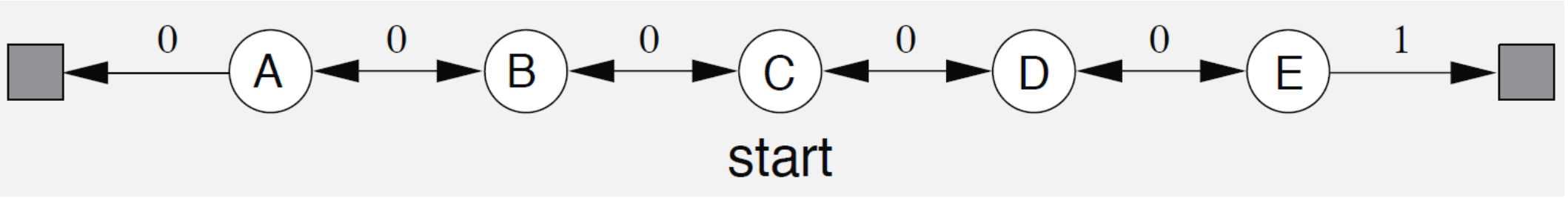
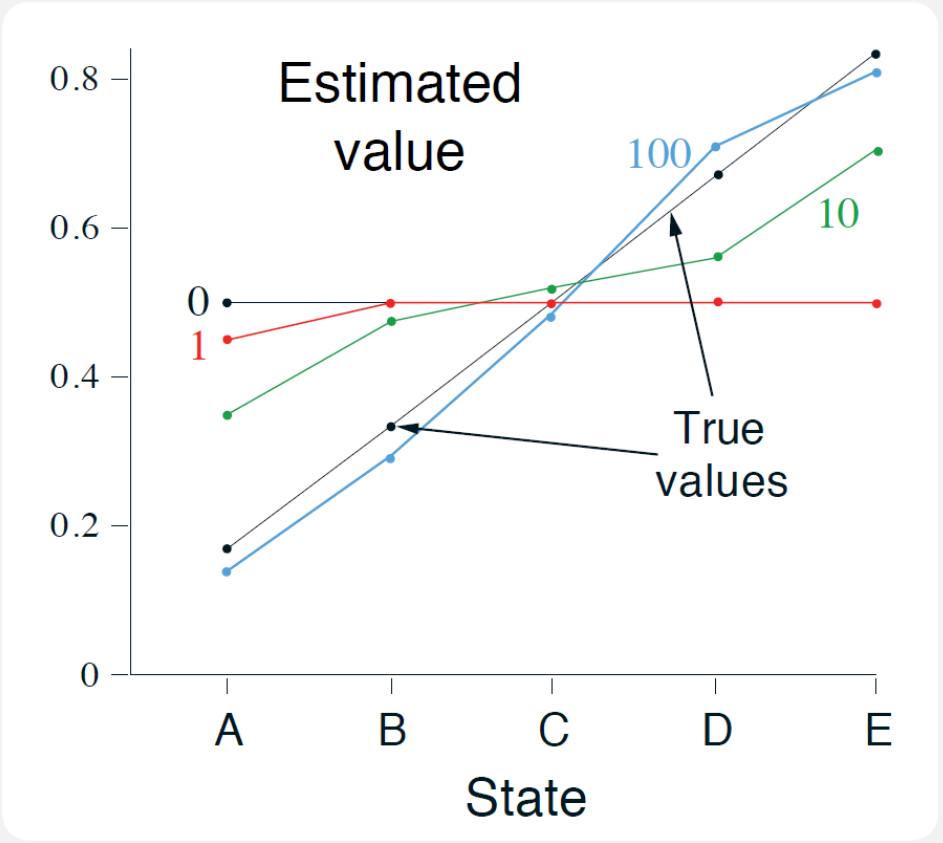
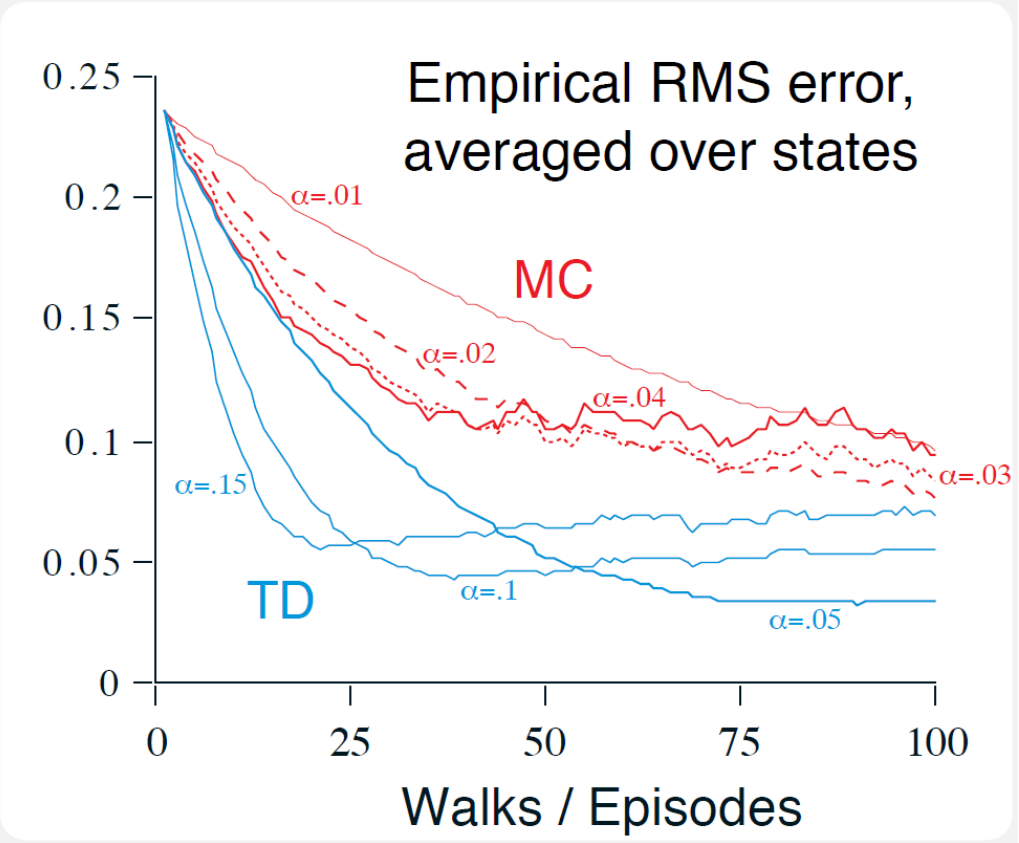


Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and n-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

Recall: Random walk







Random walk with n-step TD

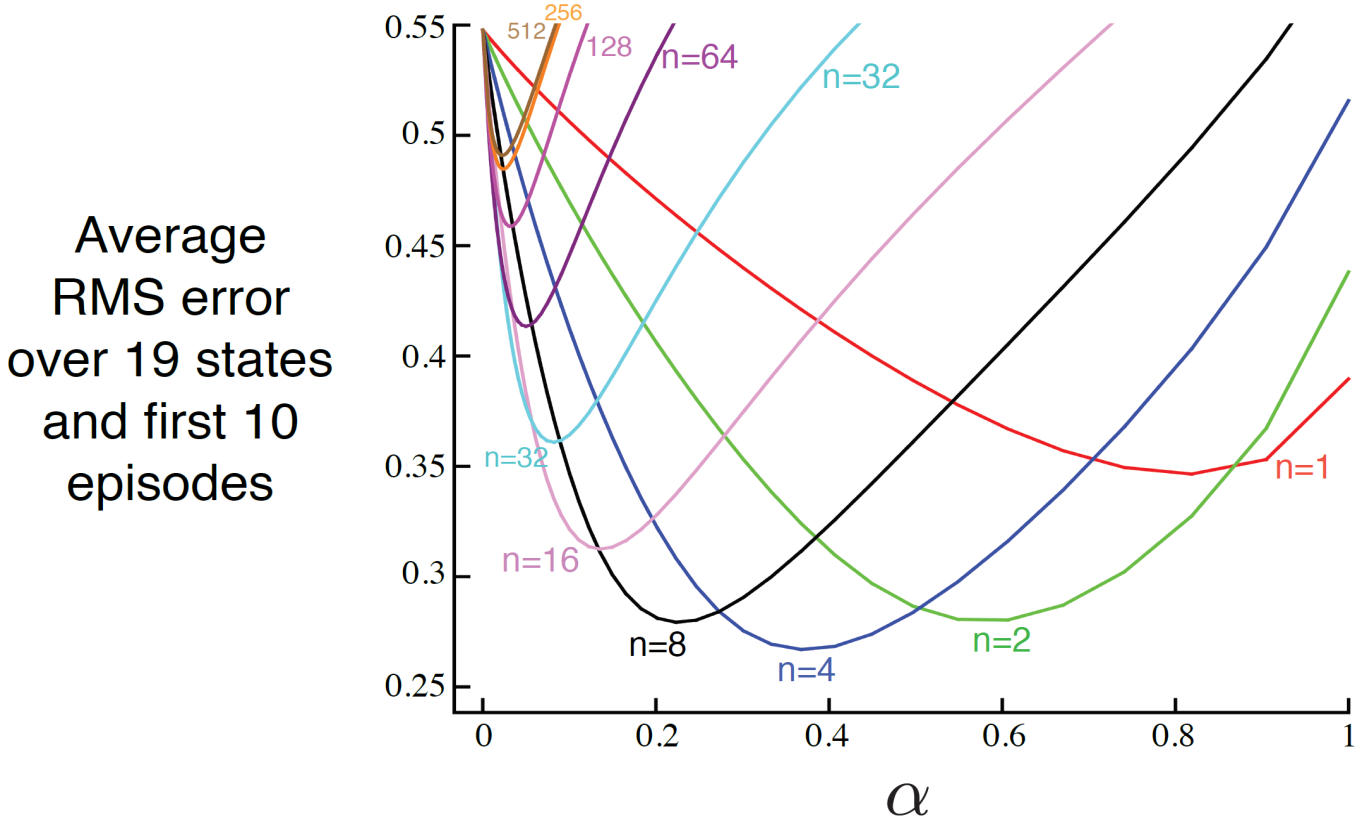


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

Properties of n-step returns

An important property of n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense. That is, the worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V_{t+n-1}:

$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \;\leq\; \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|, \tag{7.3}$$

for all $n \geq 1$. This is called the *error reduction property* of n-step returns.

Properties of n-step returns

An important property of n-step returns is that their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense. That is, the worst error of the expected n-step return is guaranteed to be less than or equal to γ^n times the worst error under V_{t+n-1}:

$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \;\leq\; \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|, \tag{7.3}$$

for all $n \geq 1$. This is called the *error reduction property* of n-step returns.

Generally, increasing *n*:

- Decreases bias
- Increases variance
- Best value depends on domain and other hyperparameters

The typical path

n-step returns:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \leq t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$G_{t:t+n} \doteq G_t \text{ if } t + n \geq T$$

Typical path from here:

- ✓ Batch average → Incremental average
- - State-value function → Action-value function
- - Prediction → Control
- - On-policy → Off-policy

The typical path

n-step returns:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \le t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$G_{t:t+n} \doteq G_t \text{ if } t+n \ge T$$

Action-values:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right], \qquad 0 \le t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \ge 1, 0 \le t < T-n$$

$$G_{t:t+n} \doteq G_t \text{ if } t+n \ge T$$

n-step SARSA

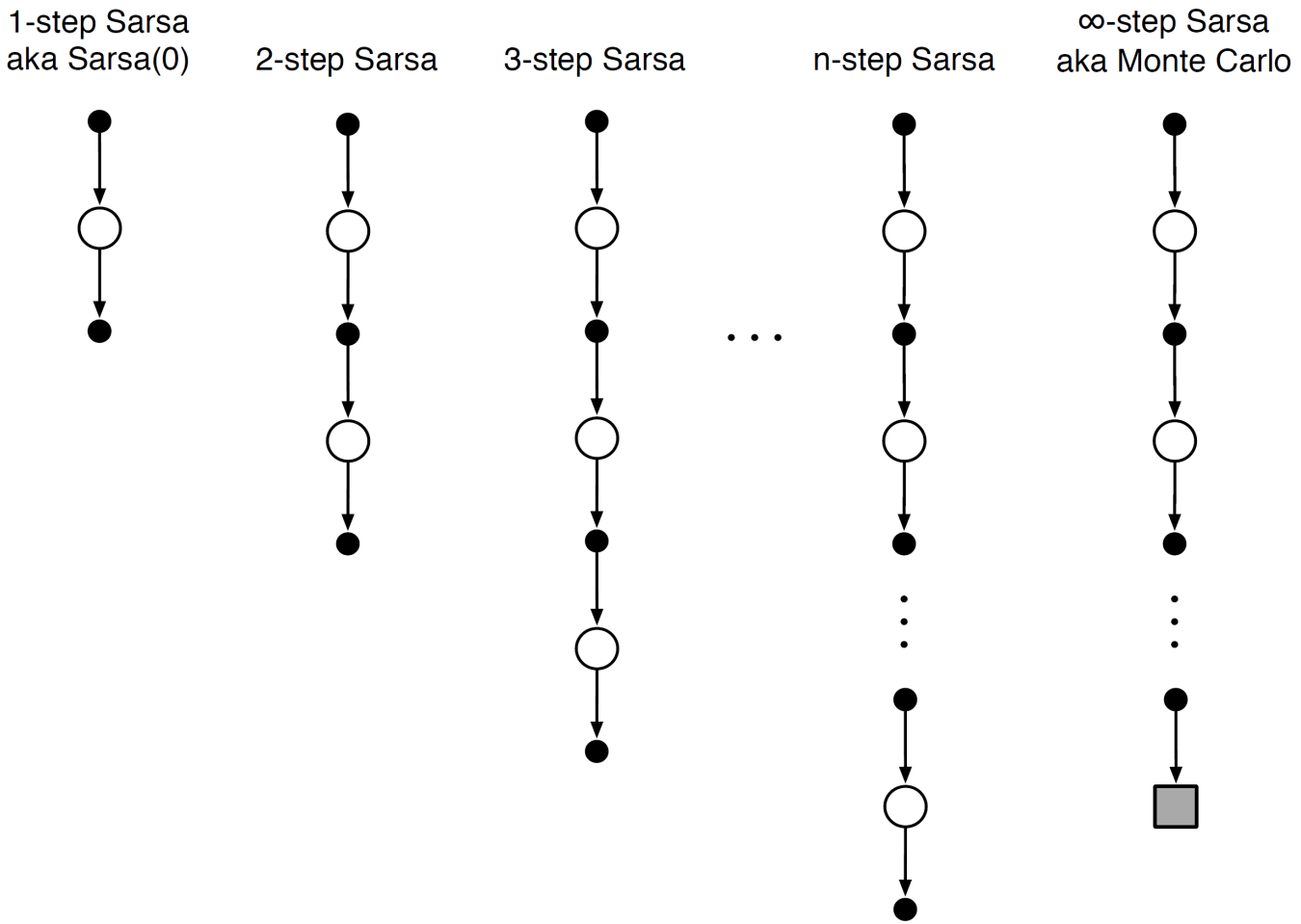


Figure 7.3: The backup diagrams for the spectrum of n-step methods for state–action values. They range from the one-step update of Sarsa(0) to the up-until-termination update of the Monte Carlo method. In between are the n-step updates, based on n steps of real rewards and the estimated value of the nth next state–action pair, all appropriately discounted. On the far right is the backup diagram for n-step Expected Sarsa.

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n + 1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | | Take action A_t
    | | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | | If S_{t+1} is terminal, then:
    | | | $T \leftarrow t + 1$
    | | else:
    | | | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | | Take action A_t
    | | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | | If S_{t+1} is terminal, then:
    | | | $T \leftarrow t+1$
    | | else:
    | | | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    | If $\tau \geq 0$:
    | | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
    | | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
    | | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
    Until $\tau = T - 1$

Example: Grid world

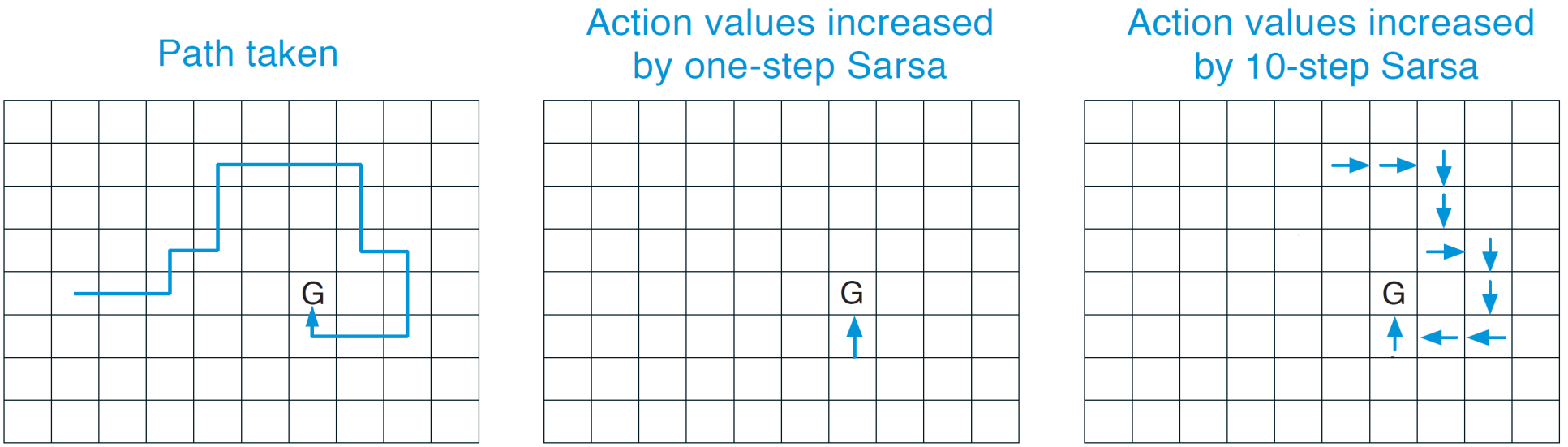


Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and n-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

The typical path

n-step returns:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \leq t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$G_{t:t+n} \doteq G_t \text{ if } t + n \geq T$$

Typical path from here:

- ✓ Batch average → Incremental average
- ✓ State-value function → Action-value function
- ✓ Prediction → Control
- - On-policy → Off-policy

  Let's leave off-policy until later …

How to choose n

Average RMS error over 19 states and first 10 episodes

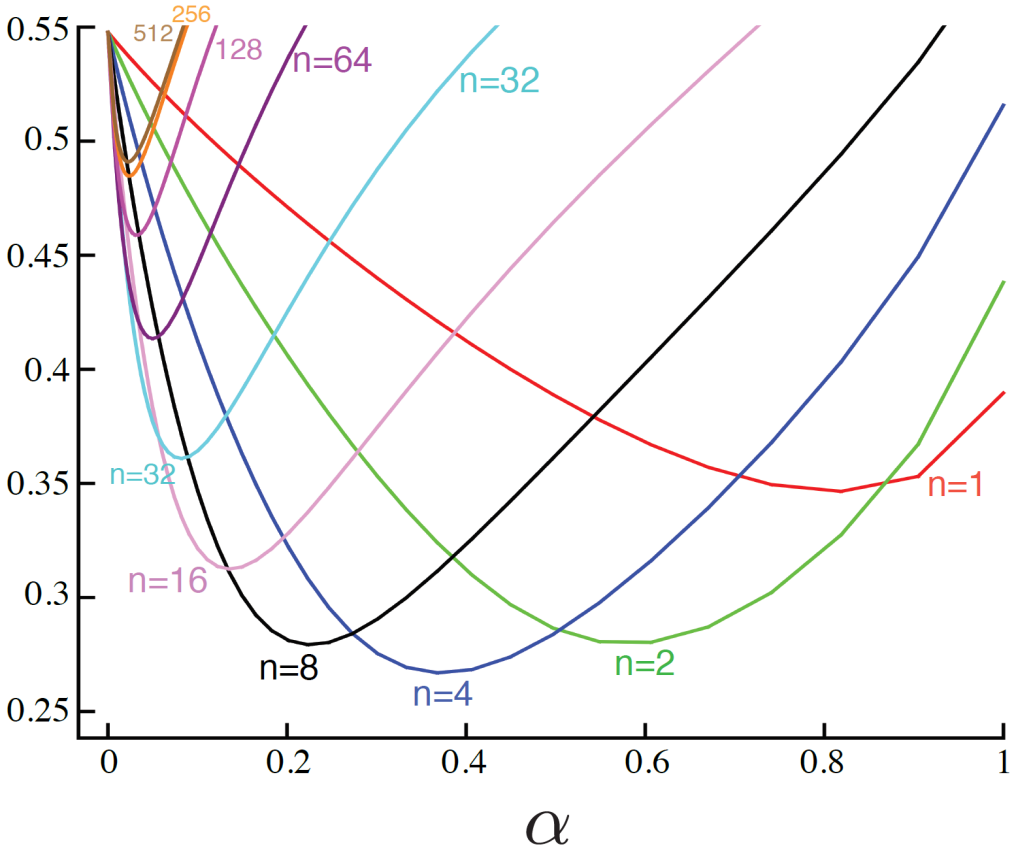


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

How to choose n

Average RMS error over 19 states and first 10 episodes

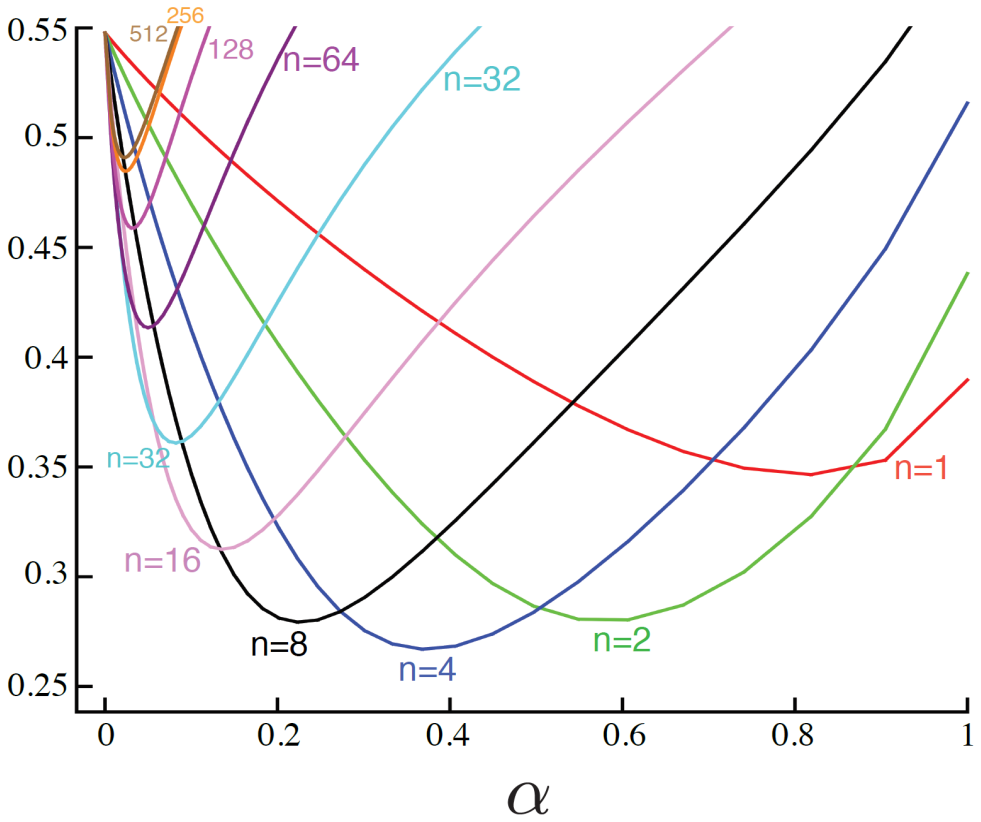


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

Maybe a single *n* is not the best idea?

How to choose n

Average RMS error over 19 states and first 10 episodes

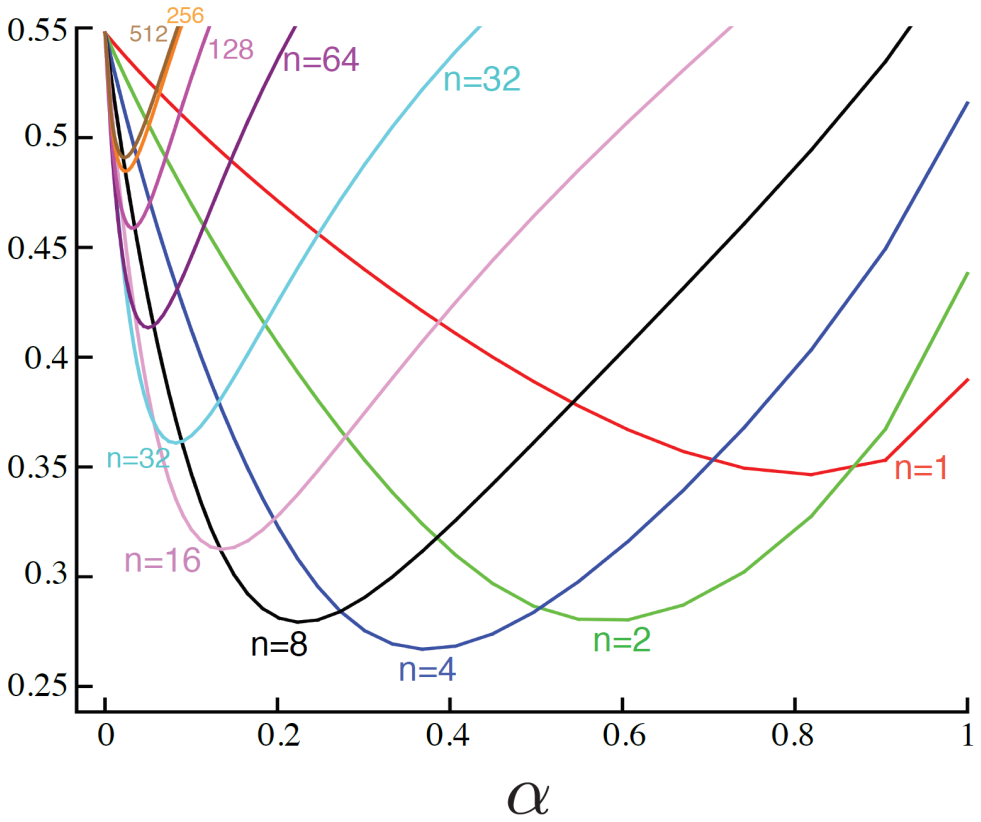


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

Maybe a single *n* is not the best idea?
How about a *mixture* over difference *n*?

λ-returns

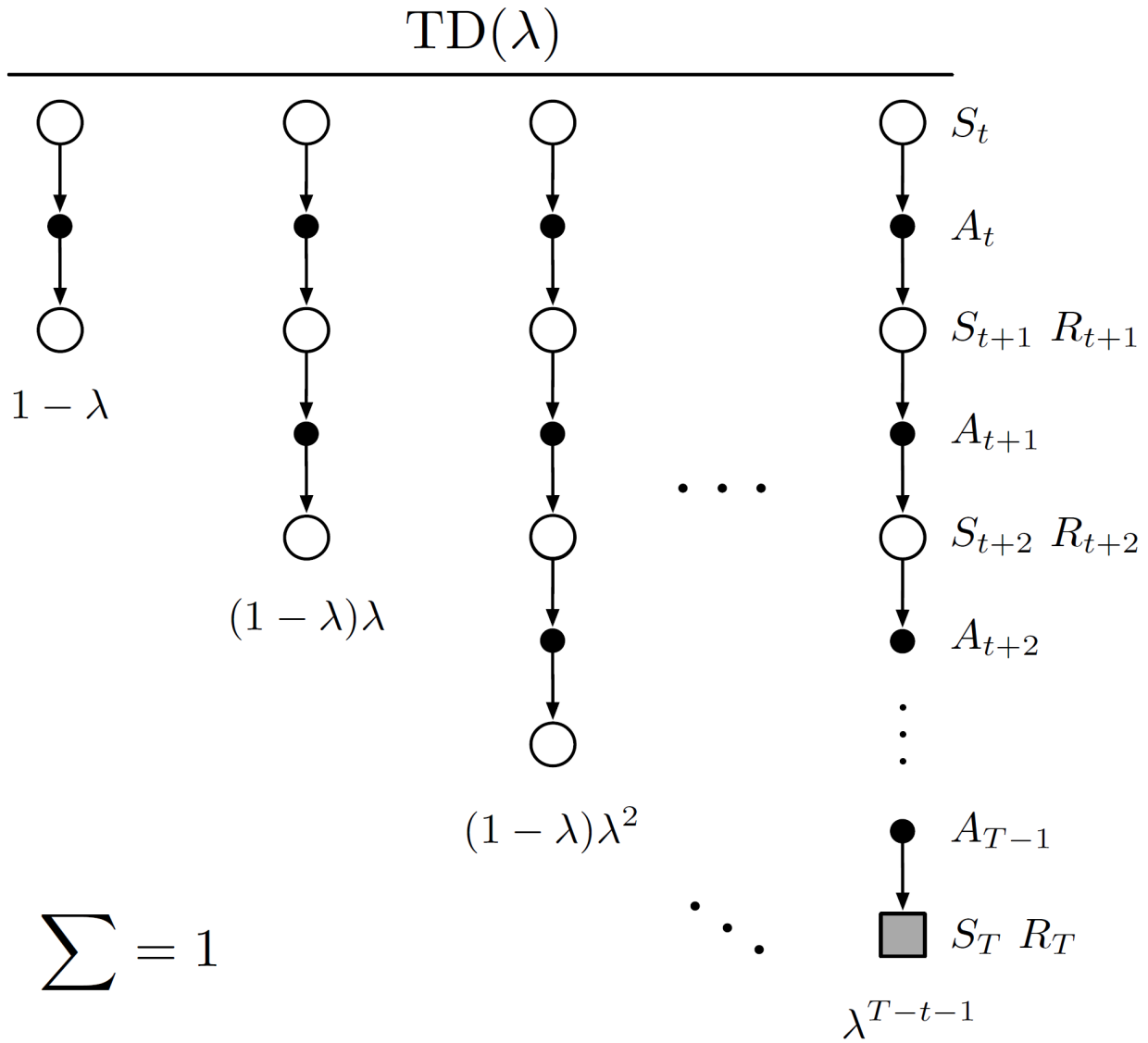


Figure 12.1: The backup digram for TD(λ). If $\lambda = 0$, then the overall update reduces to its first component, the one-step TD update, whereas if $\lambda = 1$, then the overall update reduces to its last component, the Monte Carlo update.

λ-returns

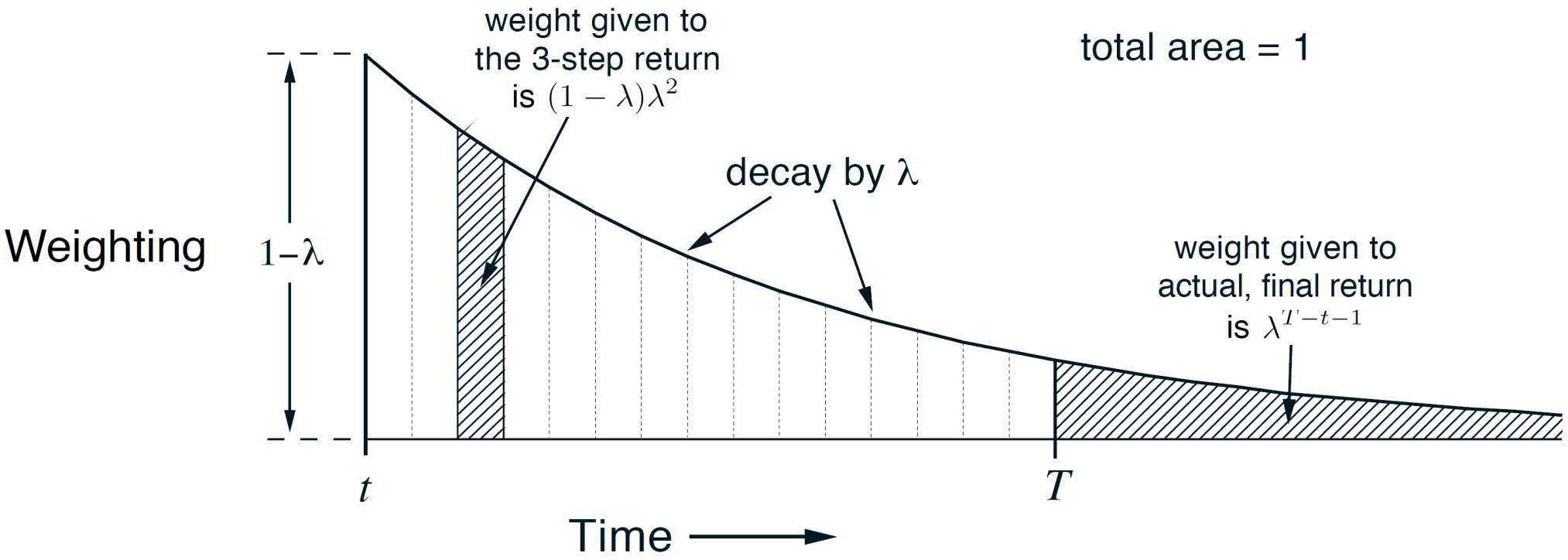


Figure 12.2: Weighting given in the λ-return to each of the n-step returns.

λ-returns

n-step returns:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \le t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$G_{t:t+n} \doteq G_t \text{ if } t+n \ge T$$

λ-return:

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

Replace one hyperparamter with another

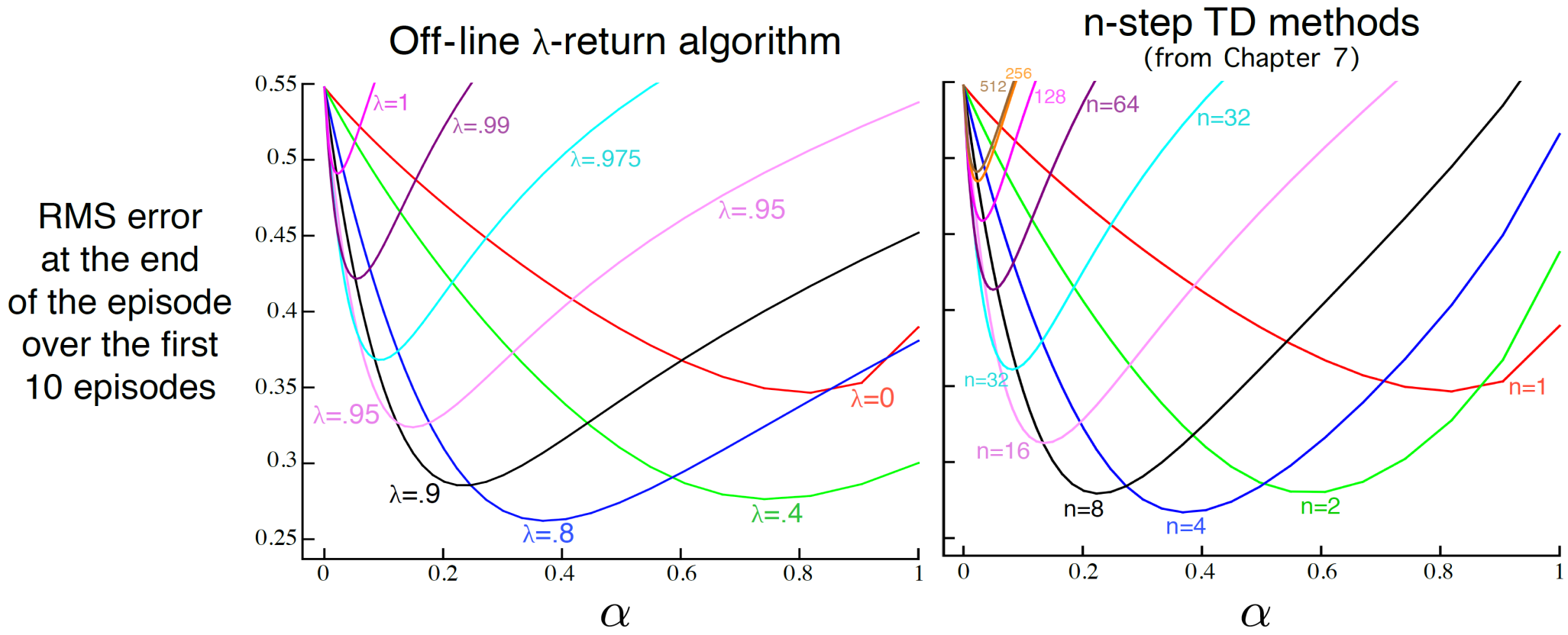


Figure 12.3: 19-state Random walk results (Example 7.1): Performance of the off-line λ-return algorithm alongside that of the n-step TD methods. In both case, intermediate values of the bootstrapping parameter (λ or n) performed best. The results with the off-line λ-return algorithm are slightly better at the best values of α and λ, and at high α.

λ-returns for action-values

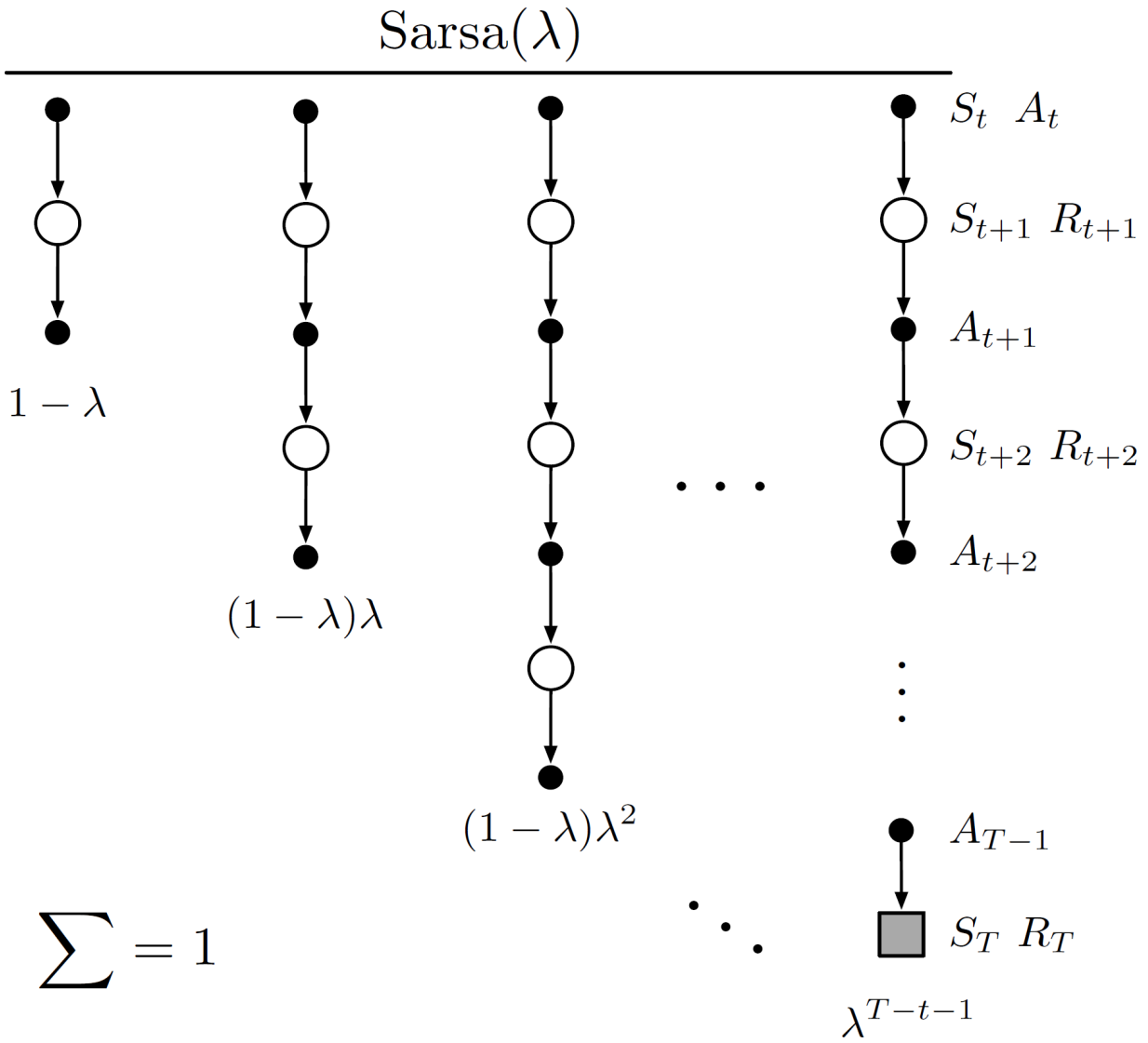
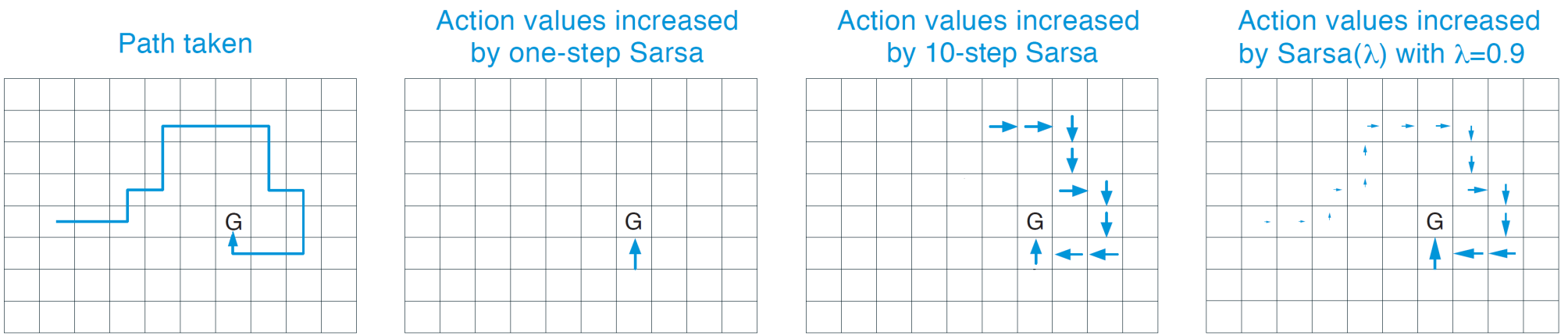


Figure 12.9: Sarsa(λ)'s backup diagram. Compare with Figure 12.1.

Example: Grid world

Example 12.1: Traces in Gridworld The use of eligibility traces can substantially increase the efficiency of control algorithms over one-step methods and even over n-step methods. The reason for this is illustrated by the gridworld example below.



The first panel shows the path taken by an agent in a single episode. The initial estimated values were zero, and all rewards were zero except for a positive reward at the goal location marked by G. The arrows in the other panels show, for various algorithms, which action-values would be increased, and by how much, upon reaching the goal. A one-step method would increment only the last action value, whereas an n-step method would equally increment the last n actions' values (assuming $\gamma = 1$), and an eligibility trace method would update all the action values up to the beginning of the episode, to different degrees, fading with recency. The fading strategy is often the best. ■

Summary: Learning targets

Monte-Carlo: $$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underline{G_t} - V(S_t)\Big]$$

TD(0): $$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underline{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)\Big]$$

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[\underline{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})} - Q(S_t, A_t)\Big]$$

Q-learning (off-policy):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[\underline{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)} - Q(S_t, A_t)\Big]$$

Expected SARSA (generalization of SARSA / Q-learning):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[\underline{R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}]} - Q(S_t, A_t)\Big]$$

Summary: Learning targets

n-step returns (generalization of TD and MC):

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \le t < T$$

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$G_{t:t+n} \doteq G_t \text{ if } t + n \ge T$$

λ-return (generalization of *n*-step returns):

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

Recall: Off-policy methods

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy b

Recall: Off-policy methods

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy *b*

Problem: Rewards *R* generated while following *b*,
but want to estimate for following π

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Recall: Off-policy methods

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy *b*

Problem: Rewards *R* generated while following *b*,
but want to estimate for following π

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Solution: Importance sampling $\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

Importance sampling strikes again

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy *b*

Problem: Rewards *R* generated while following *b*,
but want to estimate for following π

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Solution: Importance sampling $\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

Monte-Carlo: (recall)

$$\mathbb{E}[G_t|S_t = s] = v_b(s)$$

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$$

Importance sampling strikes again

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy *b*

Problem: Rewards *R* generated while following *b*,
but want to estimate for following π

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Solution: Importance sampling $\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

Monte-Carlo: (recall)

$$\mathbb{E}[G_t|S_t = s] = v_b(s)$$
$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

Importance sampling strikes again

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy *b*

Problem: Rewards *R* generated while following *b*,
but want to estimate for following π

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Solution: Importance sampling $\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

n-step:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \qquad 0 \leq t < T$$

Importance sampling strikes again

Off-policy: Learn the value of some target policy π
while executing / using experience from behavior policy *b*

Problem: Rewards *R* generated while following *b*,
but want to estimate for following π

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Solution: Importance sampling $\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

n-step: (note time-index shift for Q)

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \qquad 0 \leq t < T$$

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

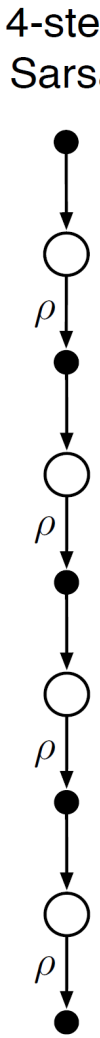
Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim b(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | | Take action A_t
    | | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | | If S_{t+1} is terminal, then:
    | | | $T \leftarrow t+1$
    | | else:
    | | | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:\tau+n})$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

Backup diagrams



What about expected SARSA?

Expected SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big]$$
$$\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big],$$

Q-learning (off-policy):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

Figure 6.4: The backup diagrams for Q-learning and Expected Sarsa.

n-step Expected SARSA

Instead of a single outcome (SARSA):

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n$$

Consider the expected outcome (expected SARSA):

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \bar{V}_{t+n-1}(S_{t+n}), \qquad t+n < T$$

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \qquad \text{for all } s \in \mathcal{S}$$

n-step Expected SARSA

Instead of a single outcome (SARSA):

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n$$

Consider the expected outcome (expected SARSA):

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} \bar{V}_{t+n-1}(S_{t+n}), \qquad t+n < T$$

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \qquad \text{for all } s \in \mathcal{S}$$

How does this apply to off-policy?

n-step Expected SARSA

Instead of a single outcome (SARSA):

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n$$

Consider the expected outcome (expected SARSA):

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} \bar{V}_{t+n-1}(S_{t+n}), \qquad t+n < T$$

$$\bar{V}_t(s) \doteq \sum_a \pi(a|s) Q_t(s,a), \qquad \text{for all } s \in \mathcal{S}$$

How does this apply to off-policy?

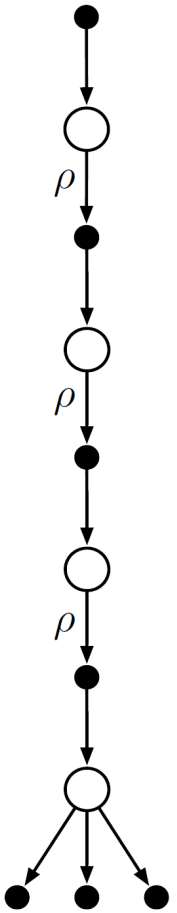
- Can do expected SARSA on or off policy
- For off-policy, *R* still comes from behavior policy, so need importance sampling
- Final time step does not need IS correction

Backup diagrams

4-step
Sarsa

$\rho_{t+1:t+n}$

4-step
Expected Sarsa



$\rho_{t+1:t+n-1}$