Temporal Difference Learning

Robert Platt
Northeastern University

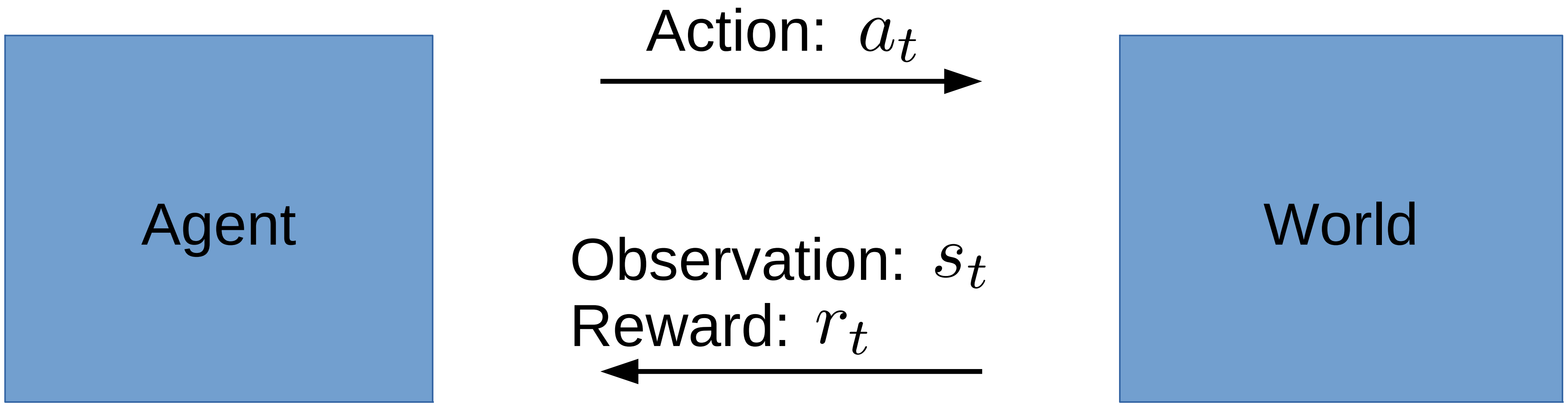
“If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning.”

– SB, Ch 6

Temporal Difference Learning

Dynamic Programming: requires a full model of the MDP
– requires knowledge of transition probabilities, reward function, state space, action space

Monte Carlo: requires just the state and action space
– does not require knowledge of transition probabilities & reward function

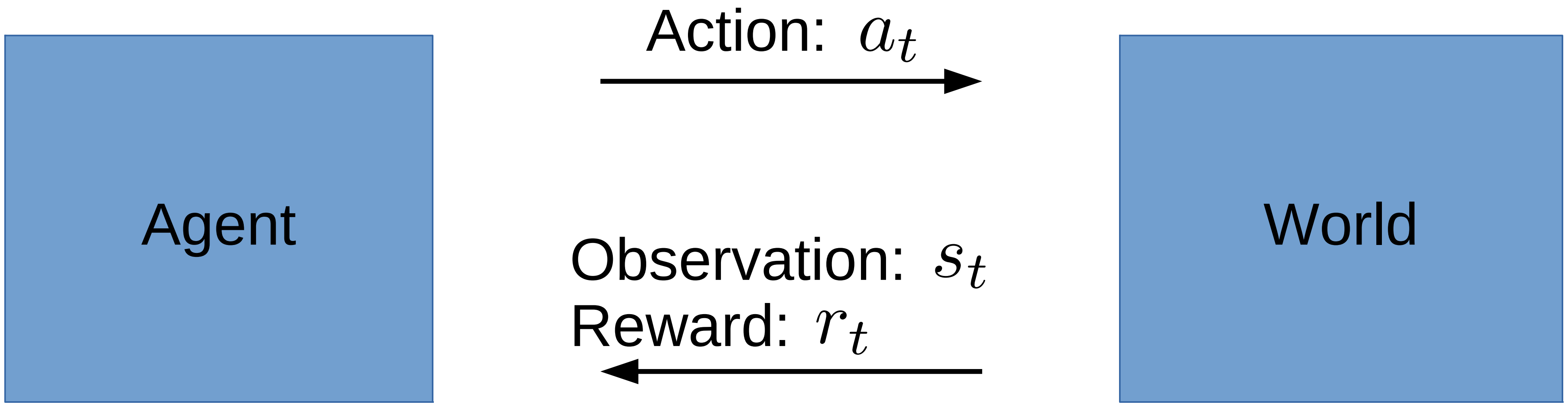


Temporal Difference Learning

Dynamic Programming: requires a full model of the MDP
– requires knowledge of transition probabilities, reward function, state space, action space

Monte Carlo: requires just the state and action space
– does not require knowledge of transition probabilities & reward function

TD Learning: requires just the state and action space
– does not require knowledge of transition probabilities & reward function



Temporal Difference Learning

Dynamic Programming:

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

or

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

Temporal Difference Learning

<u>Dynamic Programming:</u>

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

or

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

<u>Monte Carlo:</u>

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$ (SB, eqn 6.1)

Temporal Difference Learning

<u>Dynamic Programming:</u>

$$V^\pi(s_t) \leftarrow \sum_a \pi(a|s_t) \sum_{s',r} p($$

or

$$V^\pi(s_t) \leftarrow \sum_{s',r} p(s', r|s_t, \pi(s$$

Where G_t denotes total return after the first visit to s_t

<u>Monte Carlo:</u>

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$ (SB, eqn 6.1)

Temporal Difference Learning

Dynamic Programming:

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

or

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

Monte Carlo:

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$ (SB, eqn 6.1)

Temporal Difference Learning

Dynamic Programming:

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

or

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

Monte Carlo:

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$ (SB, eqn 6.1)

TD Learning:

$$\begin{aligned} V^\pi(s_t) &\leftarrow (1-\alpha)V^\pi(s_t) + \alpha[r_{t+1} + \gamma V^\pi(s_{t+1})] \\ &= V^\pi(s_t) + \alpha[r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)] \end{aligned}$$ (SB, eqn 6.2)

Temporal Difference Learning

<u>Dynamic Programming:</u>

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

or

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

<u>Monte Carlo:</u>

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha \boxed{G_t}$$ (SB, eqn 6.1)

<u>TD Learning:</u>

$$\begin{aligned} V^\pi(s_t) &\leftarrow (1-\alpha)V^\pi(s_t) + \alpha\boxed{[r_{t+1} + \gamma V^\pi(s_{t+1})]} \\ &= V^\pi(s_t) + \alpha[r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)] \end{aligned}$$ (SB, eqn 6.2)

Temporal Difference Learning

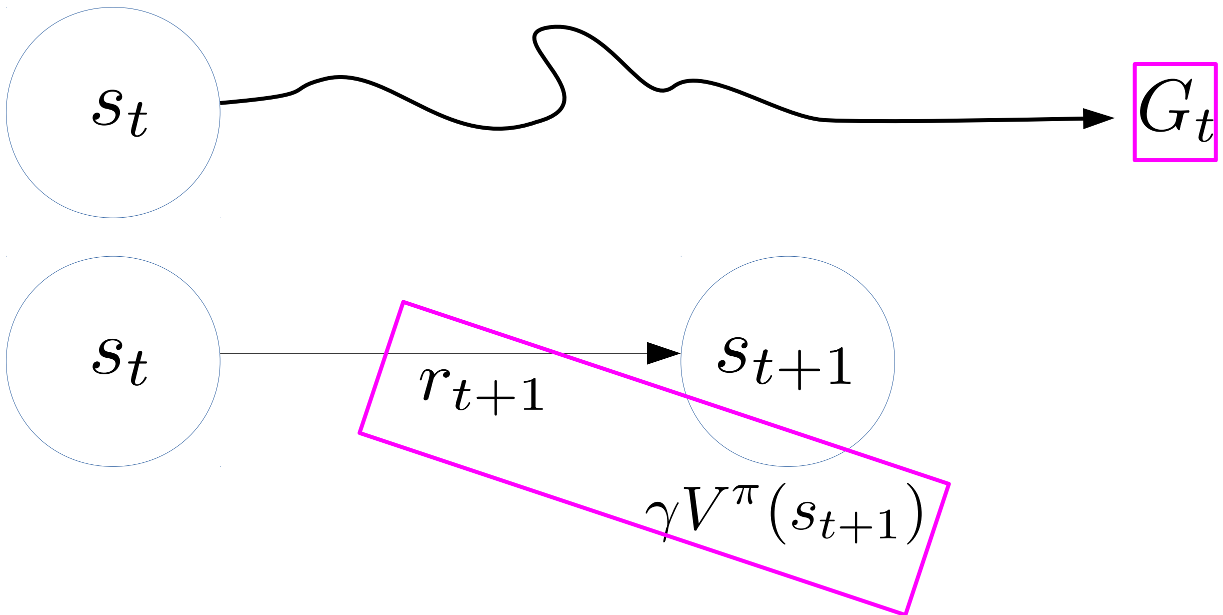
<u>Monte Carlo:</u>

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha \boxed{G_t}$$ (SB, eqn 6.1)

<u>TD Learning:</u>

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha\boxed{[r_{t+1} + \gamma V^\pi(s_{t+1})]}$$ (SB, eqn 6.2)

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



Temporal Difference Learning

Monte Carlo:

$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$ (SB, eqn 6.1)

TD Learning:

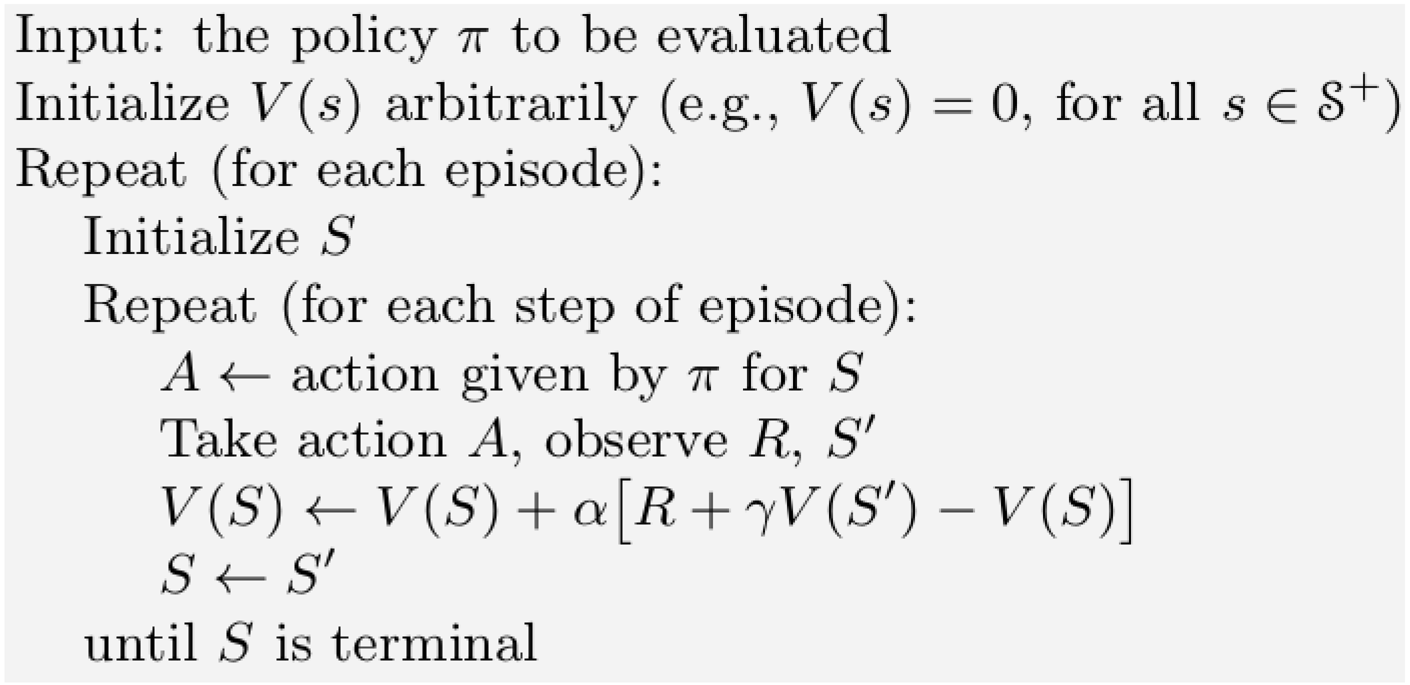
$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha[r_{t+1} + \gamma V^\pi(s_{t+1})]$$ (SB, eqn 6.2)

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

TD Error == δ_t

Temporal Difference Learning

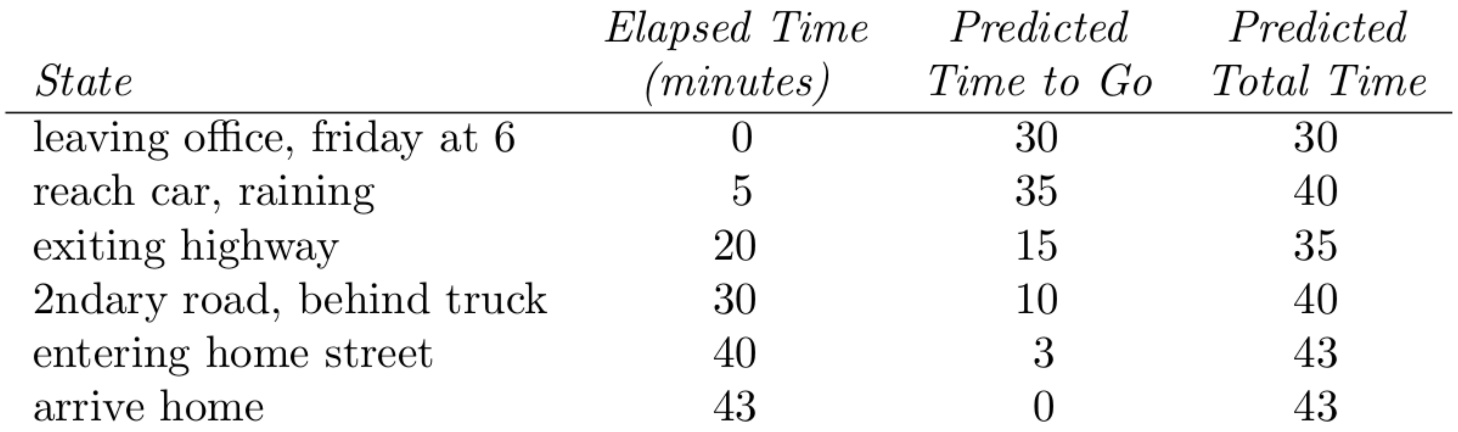
TD(0) for estimating V^{π}:



```
Input: the policy $\pi$ 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 $\pi$ 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
```

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

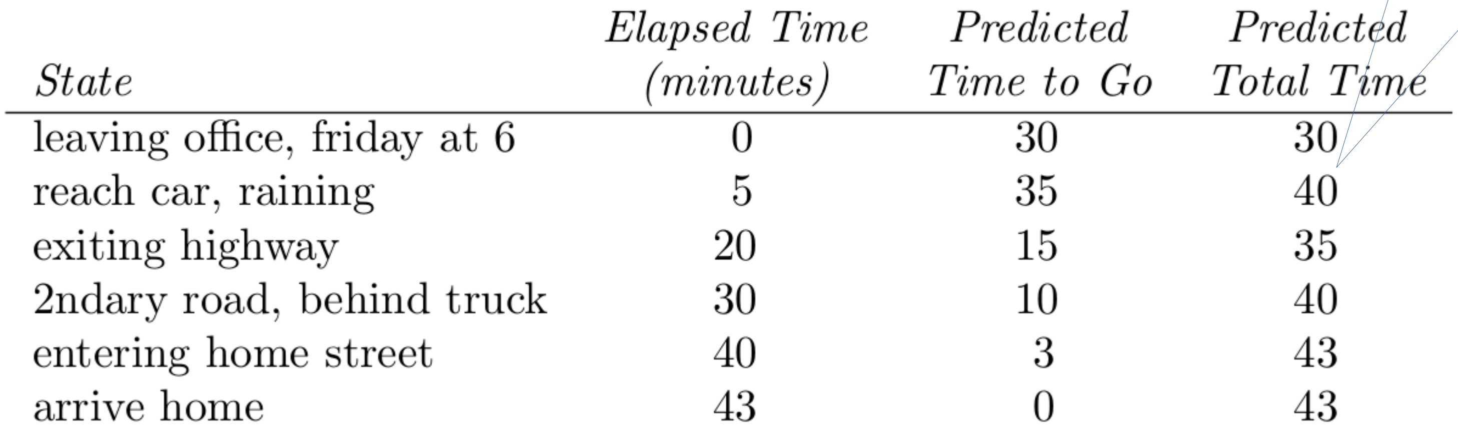
SB Example 6.1: Driving Home

Initial estimate

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

Add 10 min b/c of rain on highway



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

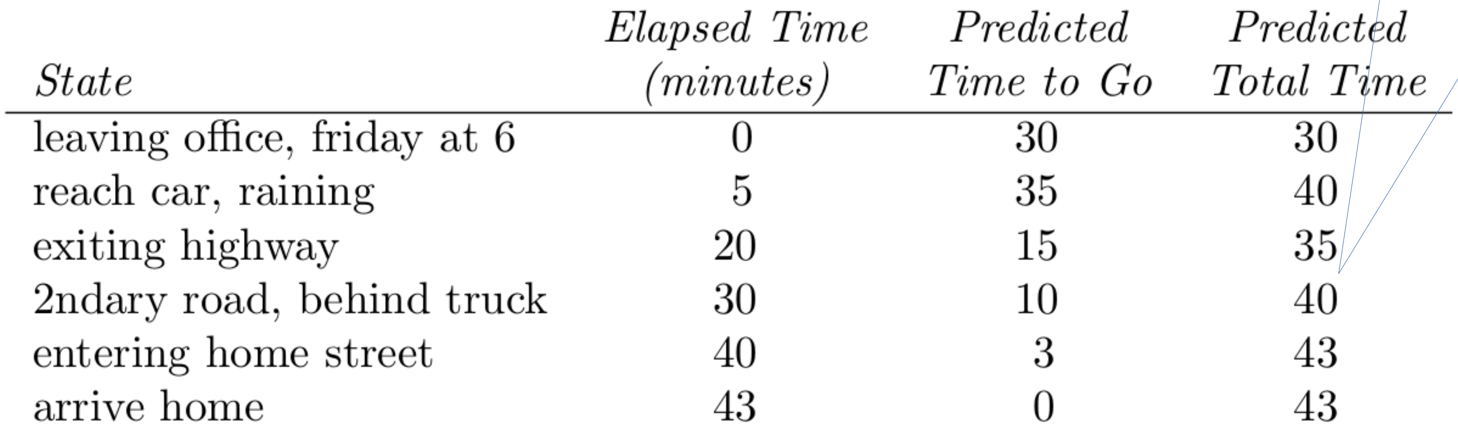
SB Example 6.1: Driving Home

Subtract 5 min b/c highway was faster than expected

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

Behind truck, add 5 min

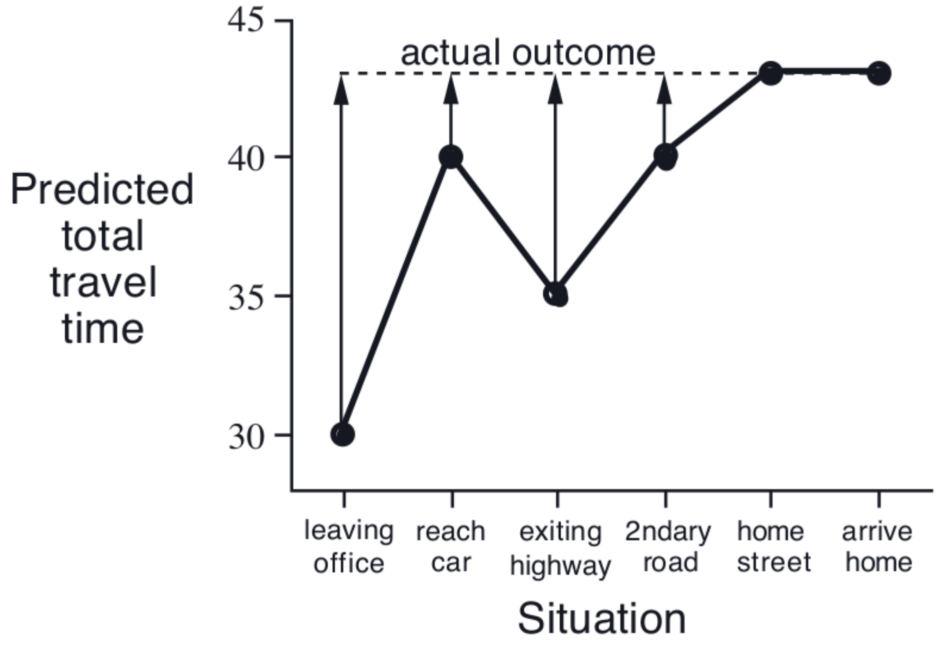


State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

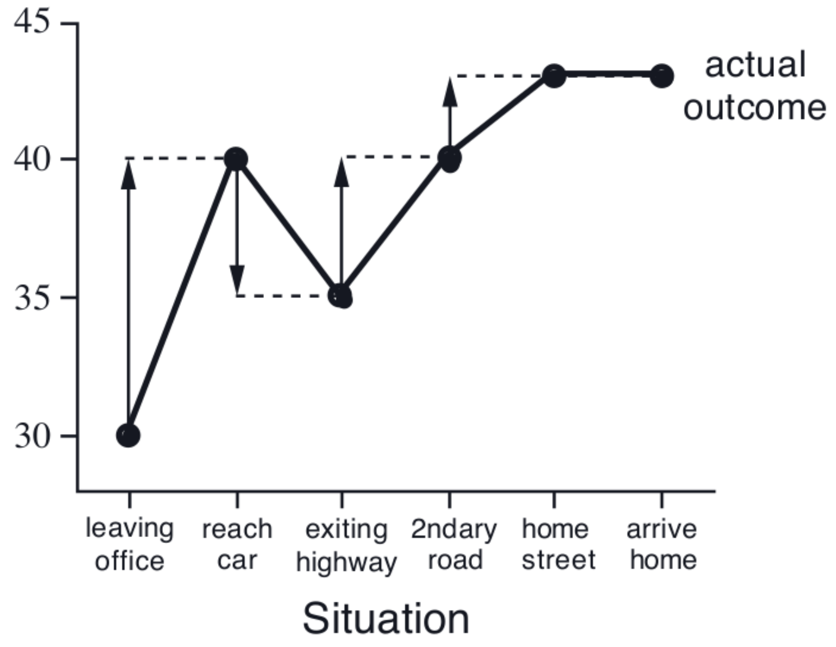
SB Example 6.1: Driving Home

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



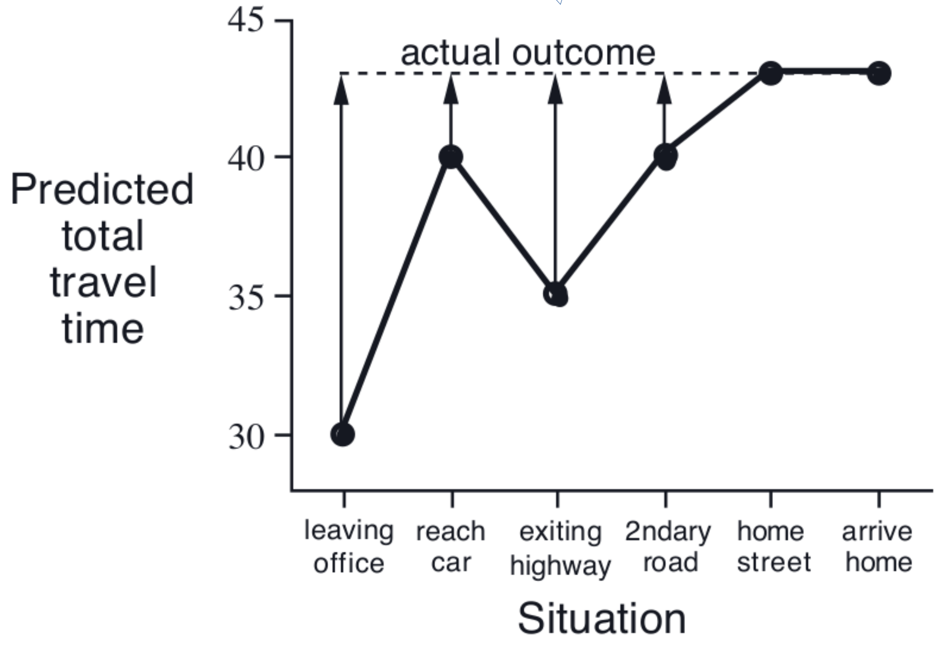
TD updates

SB Example 6.1: Driving Home

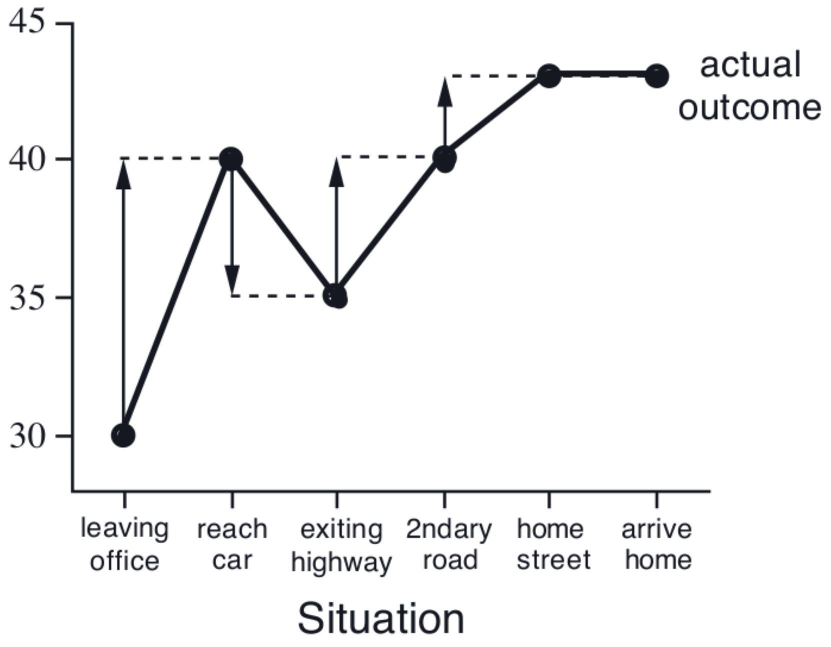
Suppose we want to estimate average time-to-go from each point along journey...

MC waits until the end before updating estimate

		redicted me to Go	Predicted Total Time
		30	30
		35	40
		15	35
2ndar nd truck	30	10	40
entering reet	40	3	43
arrive hom	43	0	43



MC updates



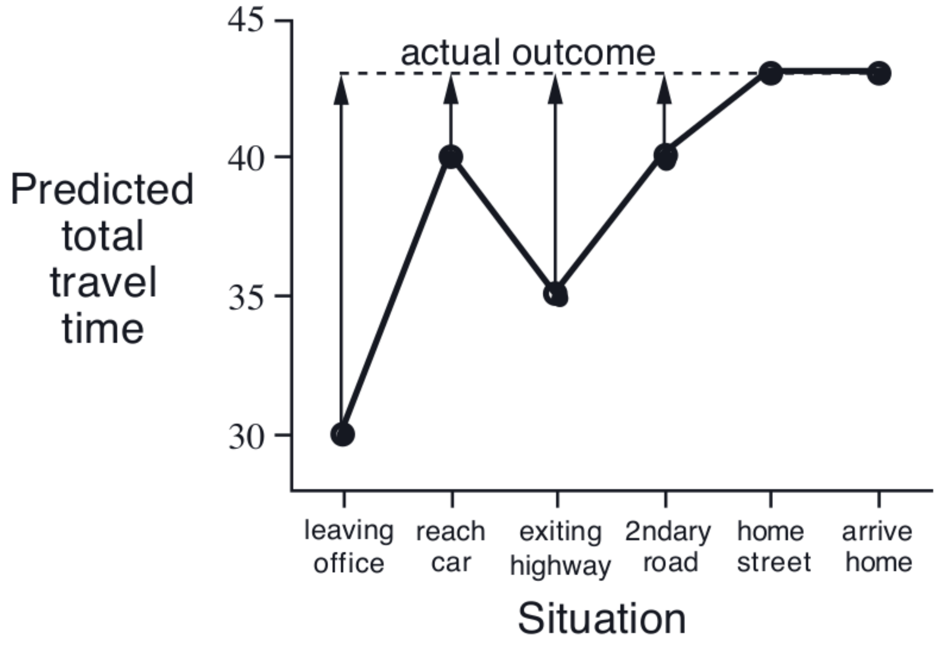
TD updates

SB Example 6.1: Driving Home

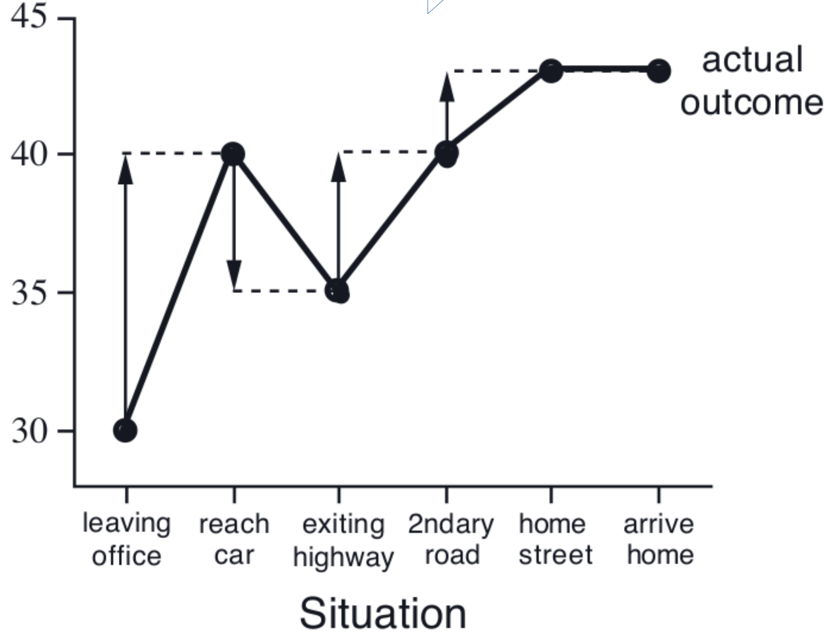
Suppose we want to estimate average time-to-go from each point along journey...

State	*Elapsed (minu*		
leaving office, friday at 6	0		
reach car, raining	5		
exiting highway	20		
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

TD updates estimate as it goes



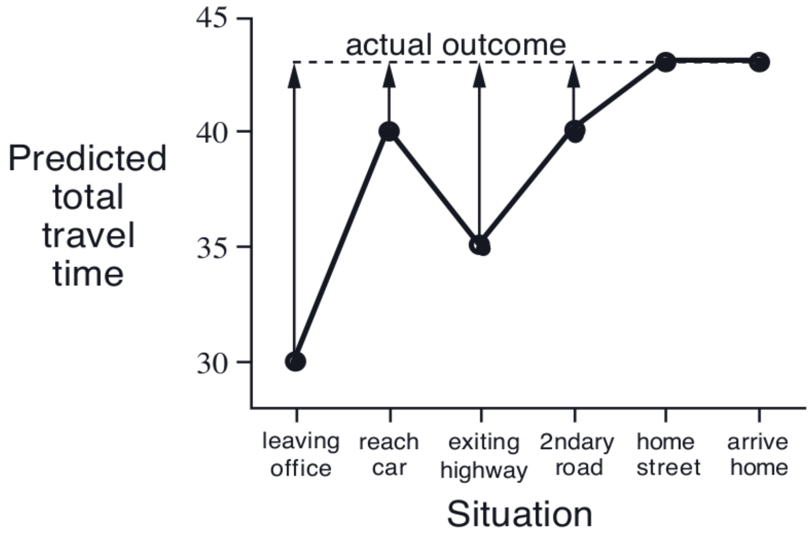
MC updates



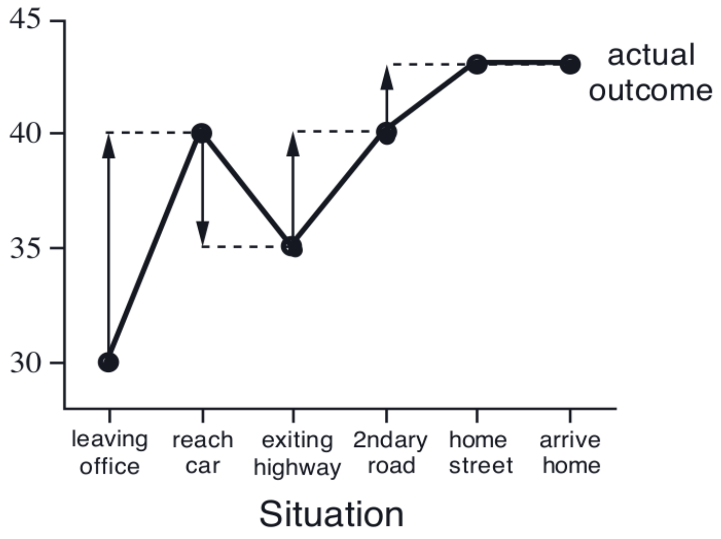
TD updates

Think-pair-share question

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

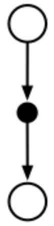


TD updates

Backup Diagrams

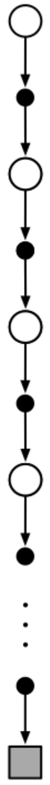
SB represents various different RL update equations pictorially as *Backup Diagrams:*

TD



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

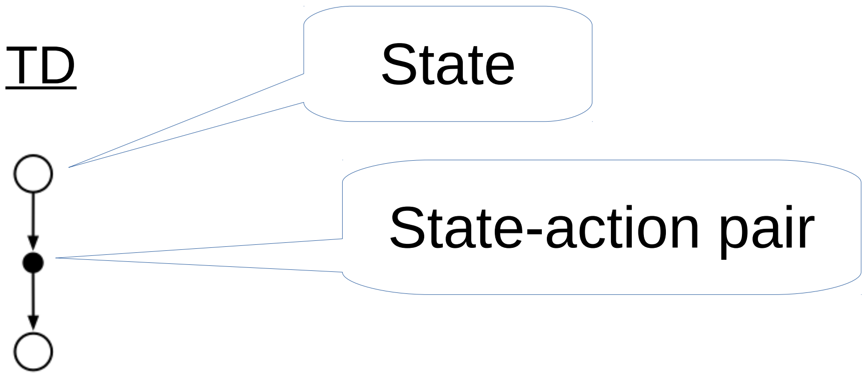
MC



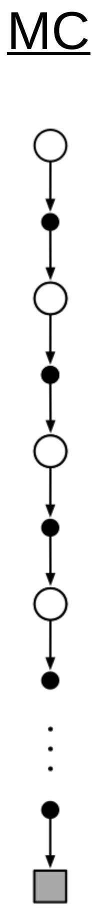
$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$

Backup Diagrams

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



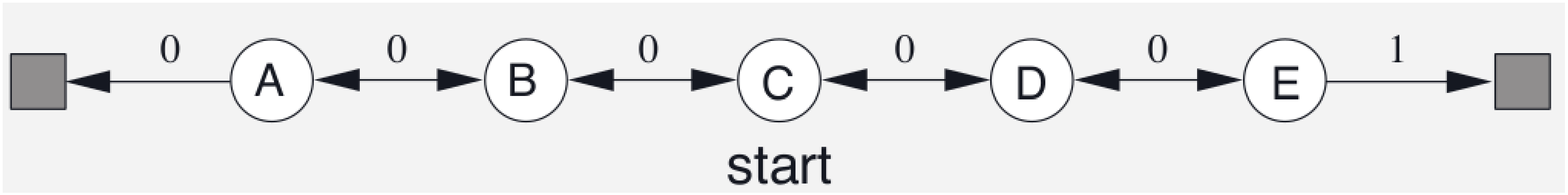
$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha[r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)]$$



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

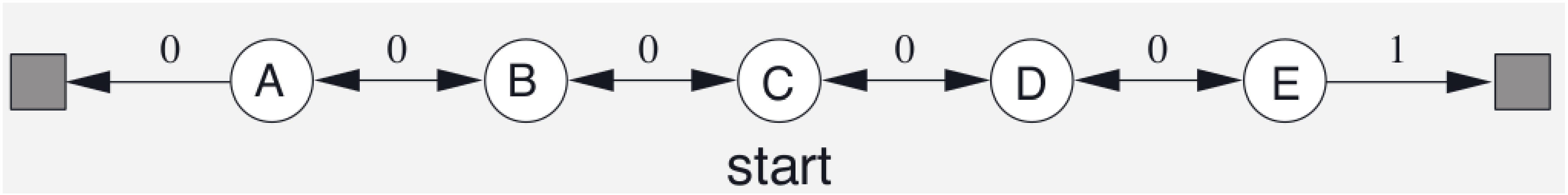
$$V^\pi(s_t) \leftarrow (1-\alpha)V^\pi(s_t) + \alpha G_t$$

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

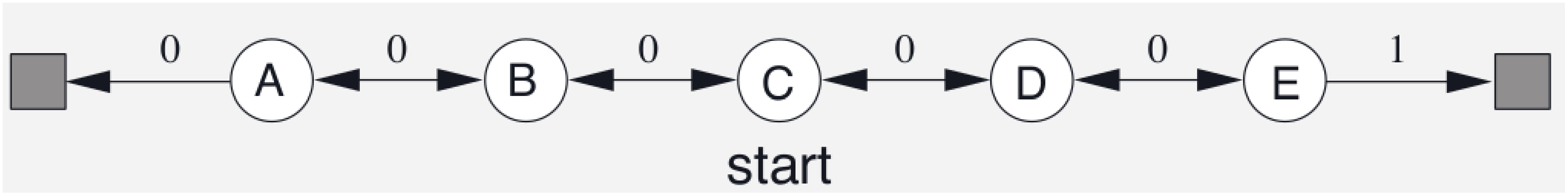


– This is a Markov Reward Process (MDP with no actions)
– Episodes start in state C
– On each time step, there is an equal probability of a left or right transition
– +1 reward at the far right, 0 reward elsewhere
– discount factor of 1
– the true values of the states are: $V(A) = \frac{1}{6}, V(B) =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

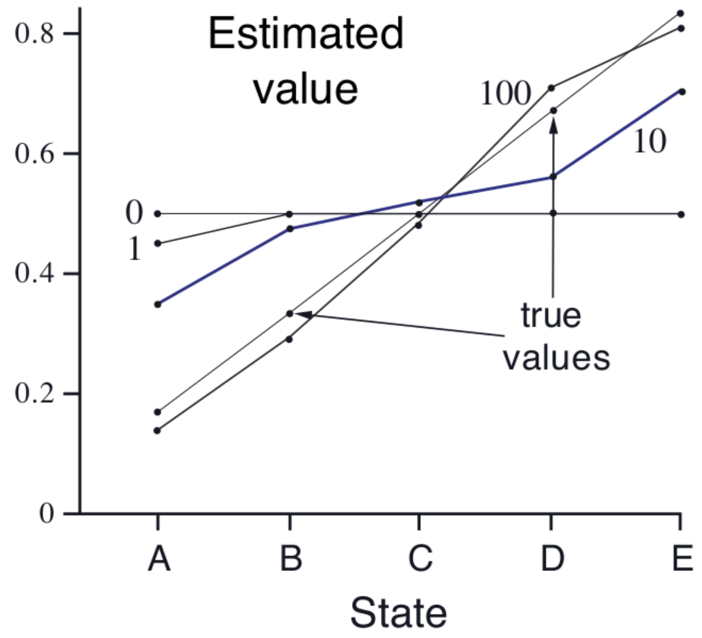
1. express the relationship between the value of a state and its neighbors in the simplest form

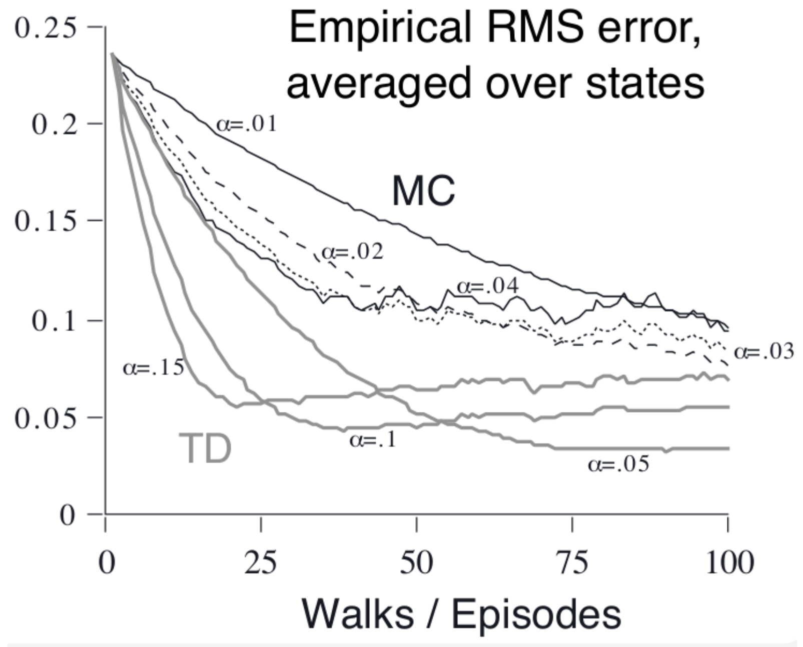
2. say how you could calculate the value of each/all states in closed form

SB Example 6.2: Random Walk

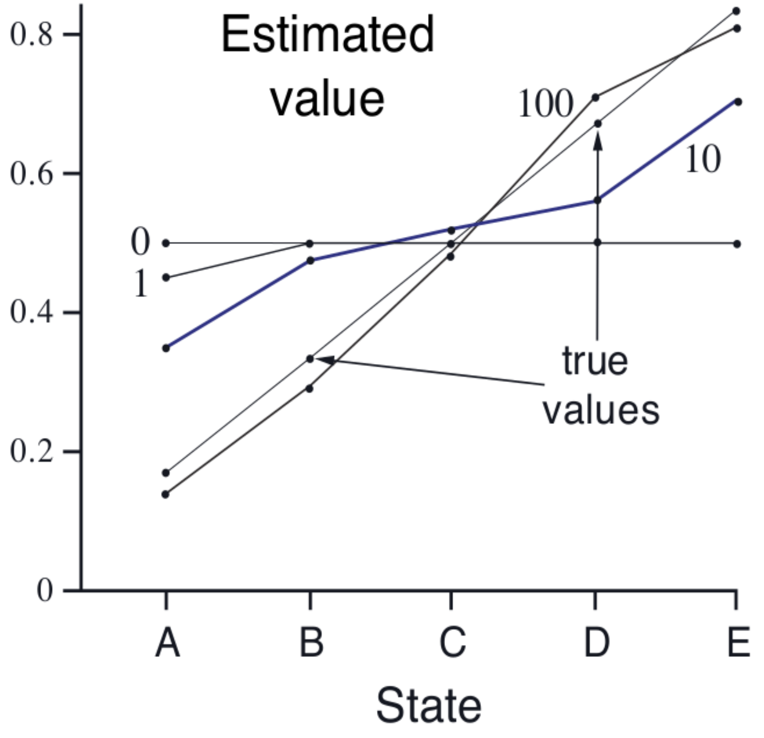


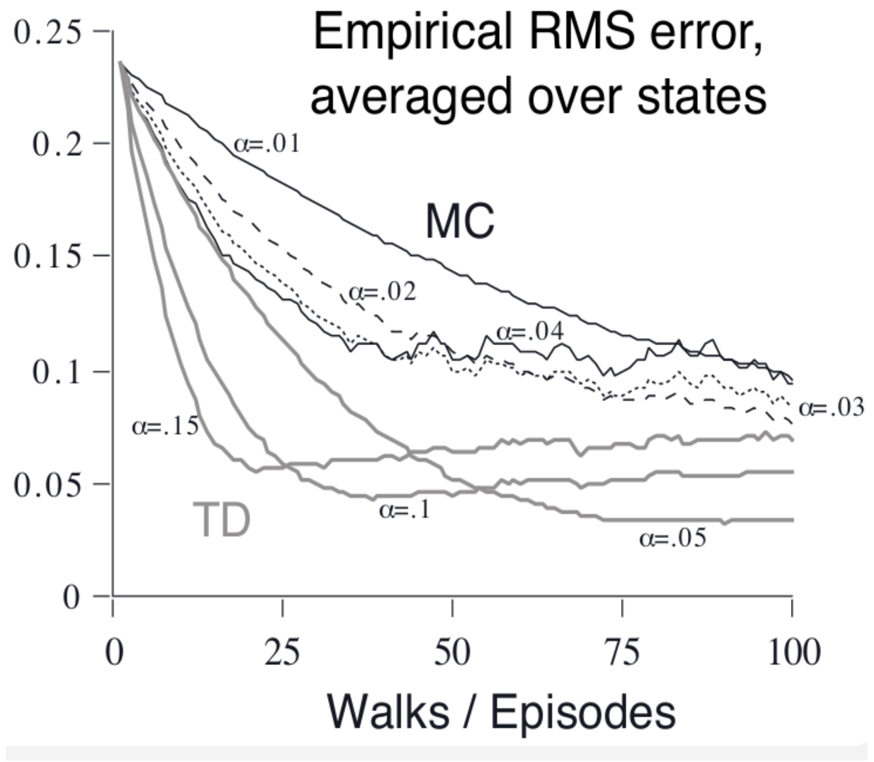
- This is a Markov Reward Process (MDP with no actions)
- Episodes start in state C
- On each time step, there is an equal probability of a left or right transition
- +1 reward at the far right, 0 reward elsewhere
- discount factor of 1
- the true values of the states are: $V(A) = \frac{1}{6}, V(B) =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





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 $V(\mathsf{A})$. 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?

Pro/Con List: TD, MC, DP

DP

Pro	Con
Efficient	Requires full model
Complete	

MC

Pro	Con
Simple	Slower than TD
Complete	High variance

TD

Pro	Con
Faster than MC	
Complete	
Low variance	

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

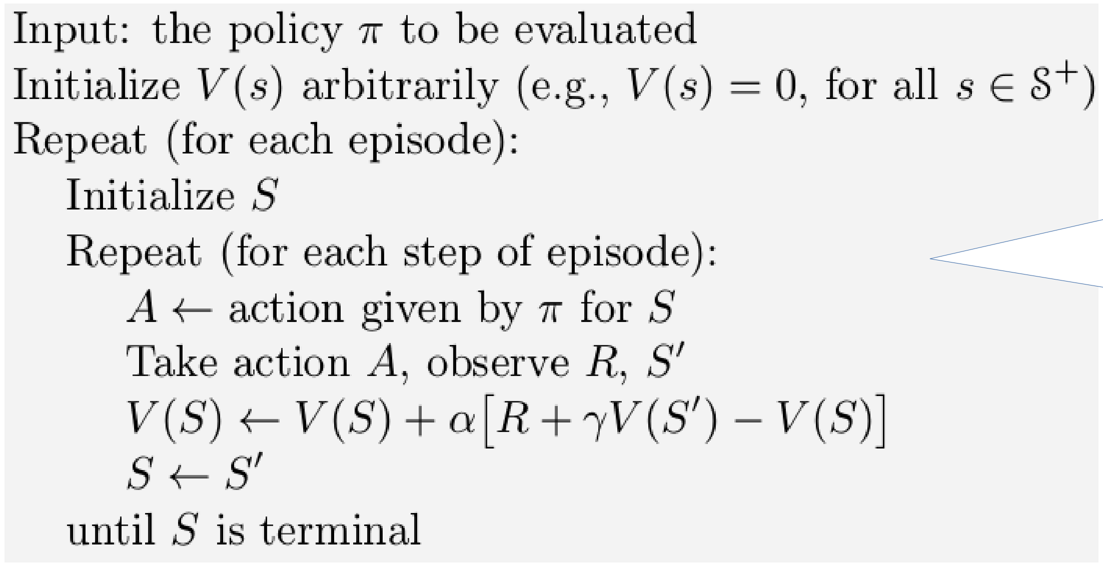
– converges exactly with a step size parameter that decreases in size

Convergence/correctness of TD(0)

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

On-line TD(0)

TD(0) for estimating V^{π}:



```
Input: the policy $\pi$ 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 $\pi$ 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

<u>Batch updating:</u>

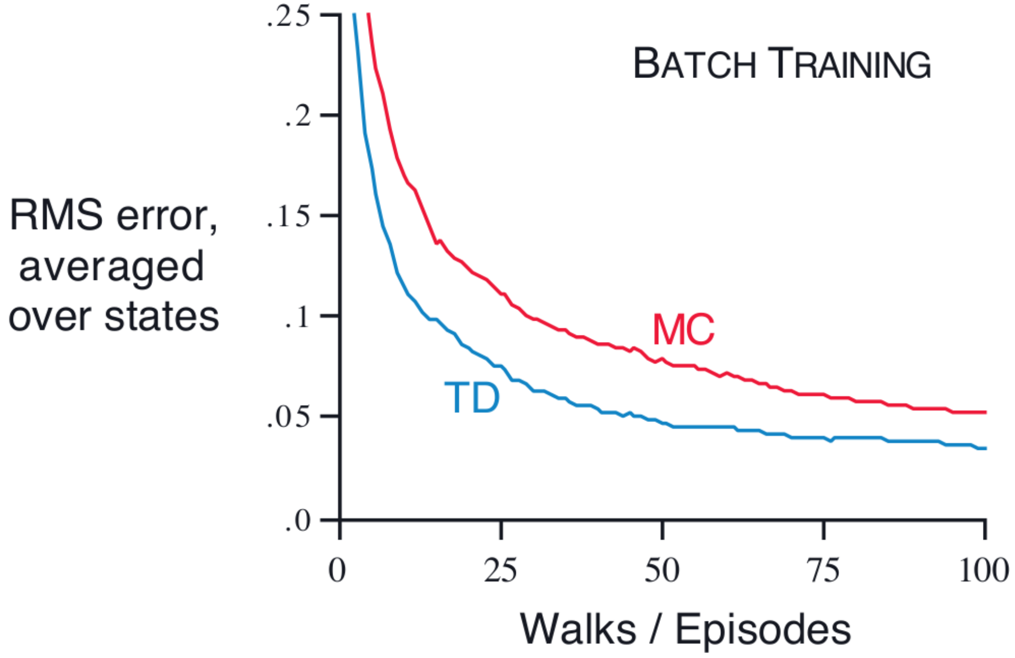
```
Collect a dataset $\mathbb{D}$ of experience (somehow)
Initialize V arbitrarily
Repeat until V converged:
    $V' = V$
    For all $(s, a, s', r) \in \mathbb{D}$:
        $V'(s) = V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $V = V'$
```

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

TD(0)/MC comparison

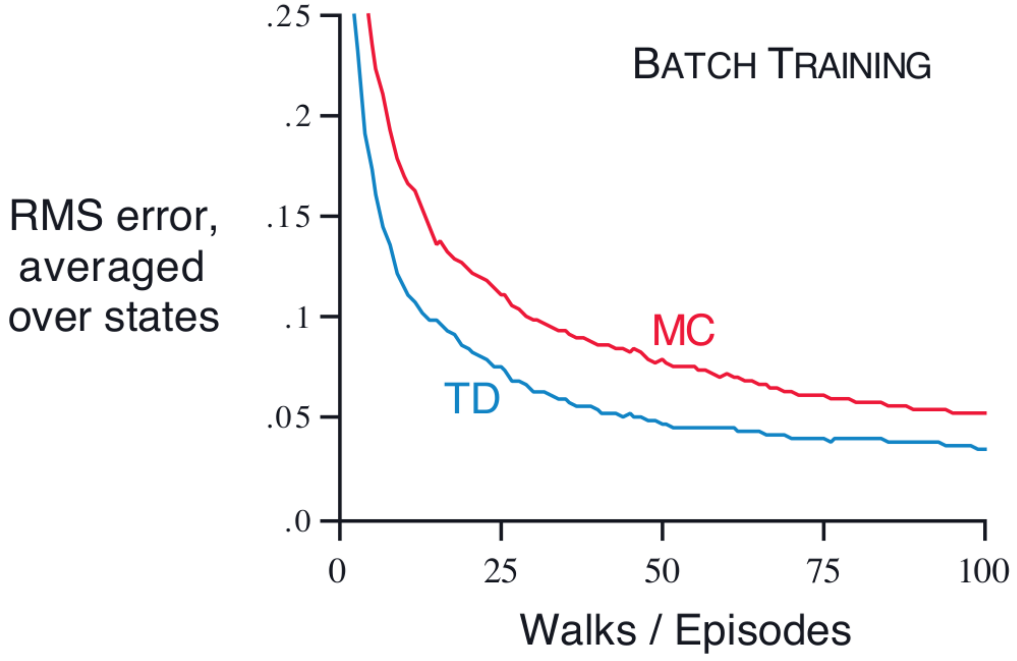
Batch TD(0) and batch MC both converge for sufficiently small step size
– but they converge to different answers!



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

Given: an undiscounted Markov reward process with two states: A, B

The following 4 episodes:

A,0,B,1
A,2
B,0,A,2
B,0,A,0,B,1

Calculate:

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

2. the maximum likelihood model of this Markov reward process. Sketch the state-transition diagram

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

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: $V^{\pi}(s)$

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

State-value fn

Action-value fn

Update rule for TD(0):

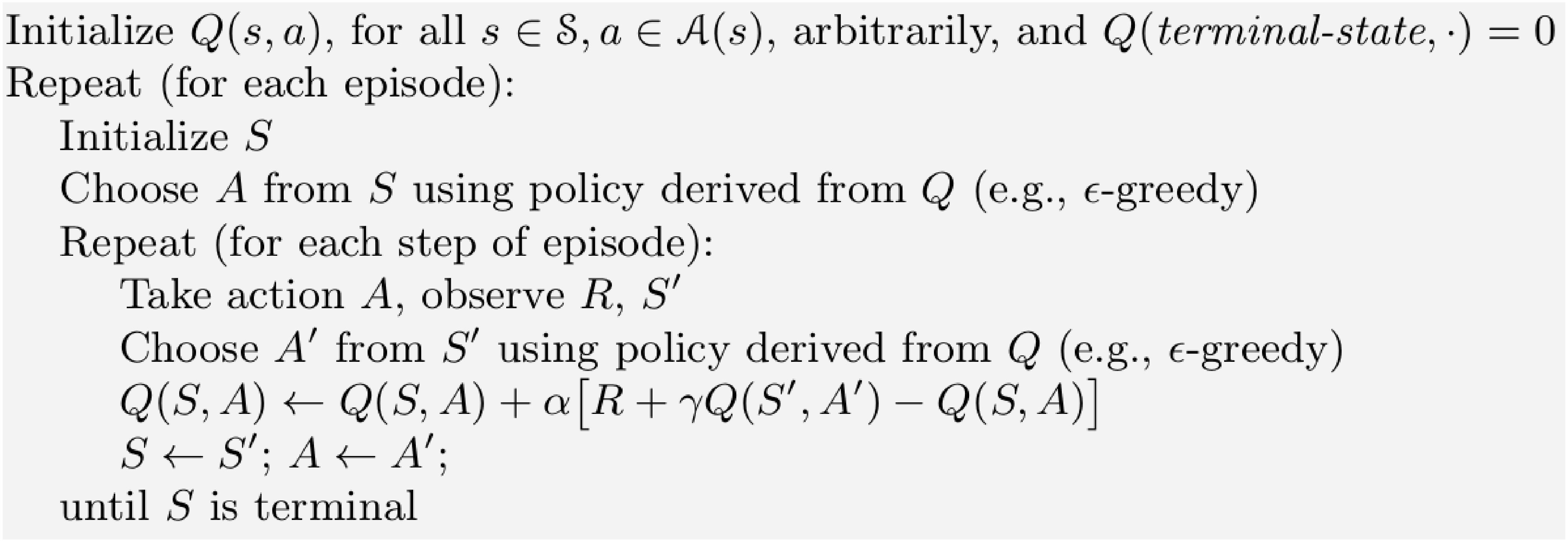
$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha[r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)]$$

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:



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

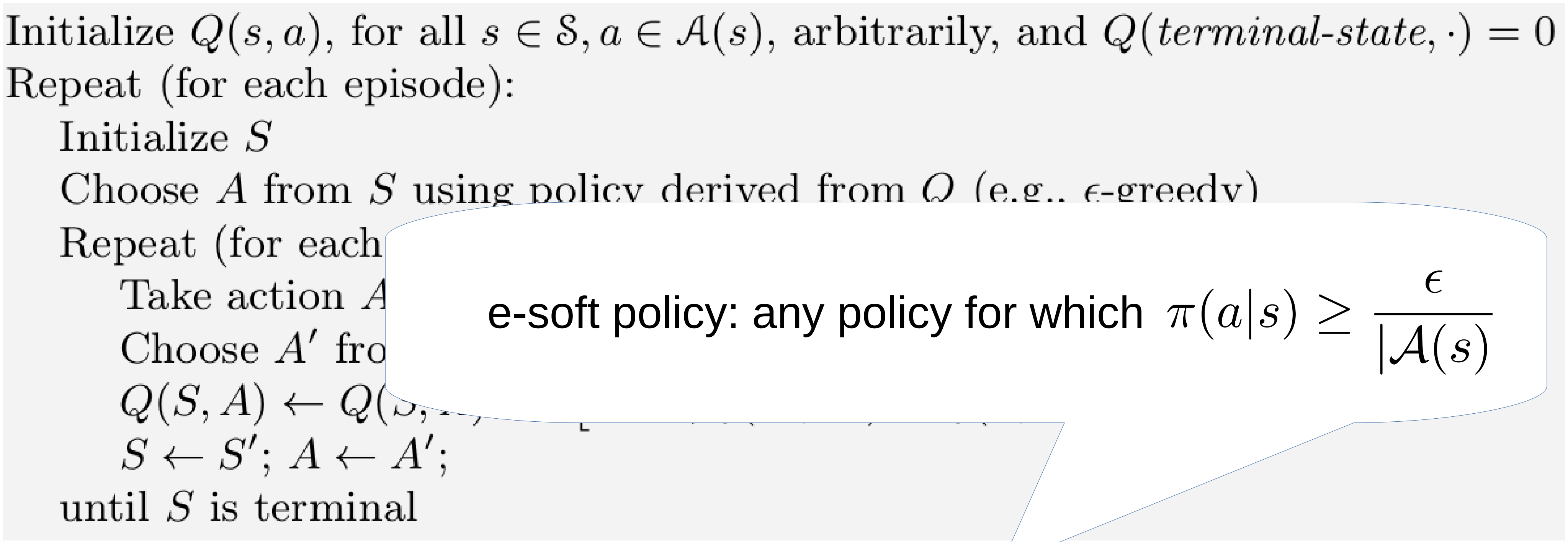
```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-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 e-greedy) w/ e>0

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

SARSA: TD Learning for Control

SARSA:



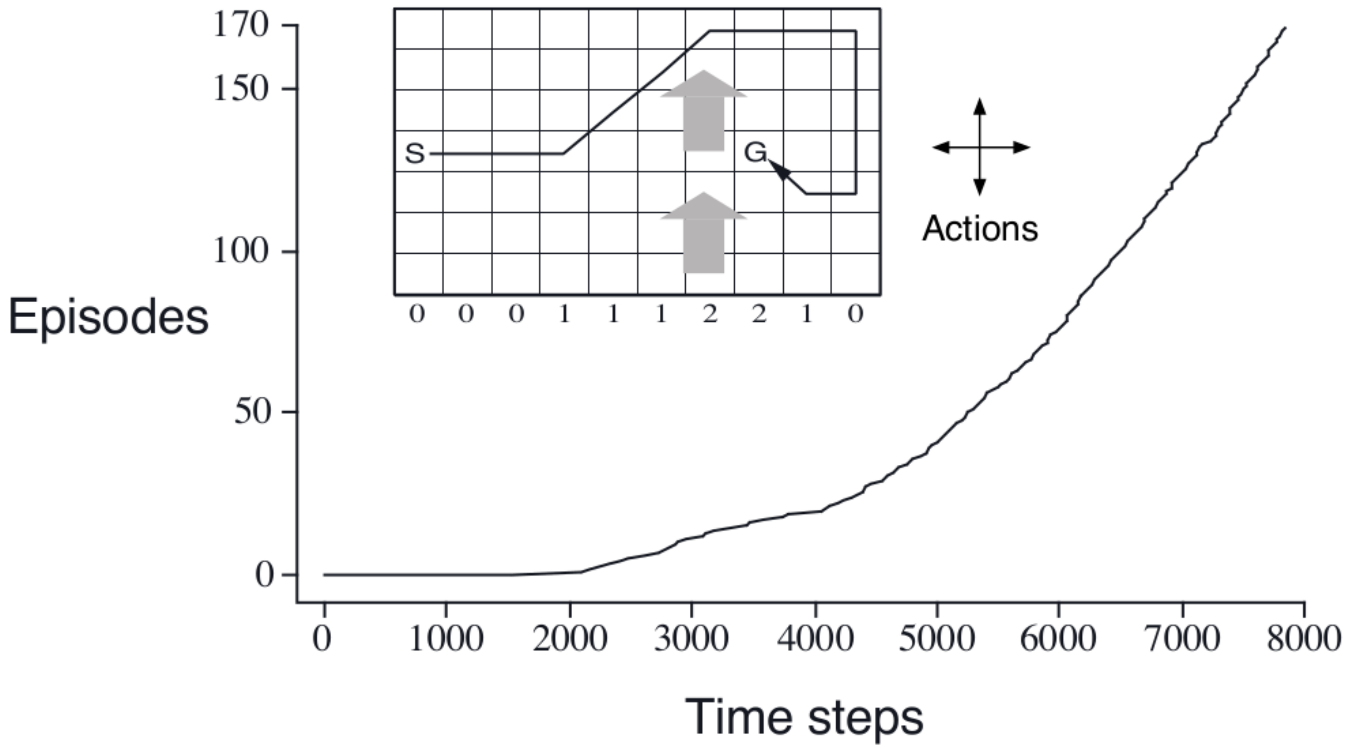
```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each
        Take action $A$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

e-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 e-greedy) w/ e>0

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

SARSA Example: Windy Gridworld



– reward = -1 for all transitions until termination at goal state
– undiscounted, deterministic transitions
– episodes only terminate at goal state

– this would be hard to solve using MC b/c episodes are very long
– optimal path length from start to goal: 15 time steps
– average path length 17 time steps (why is this longer?)

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:

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

Update rule for Q-Learning:

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

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:

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

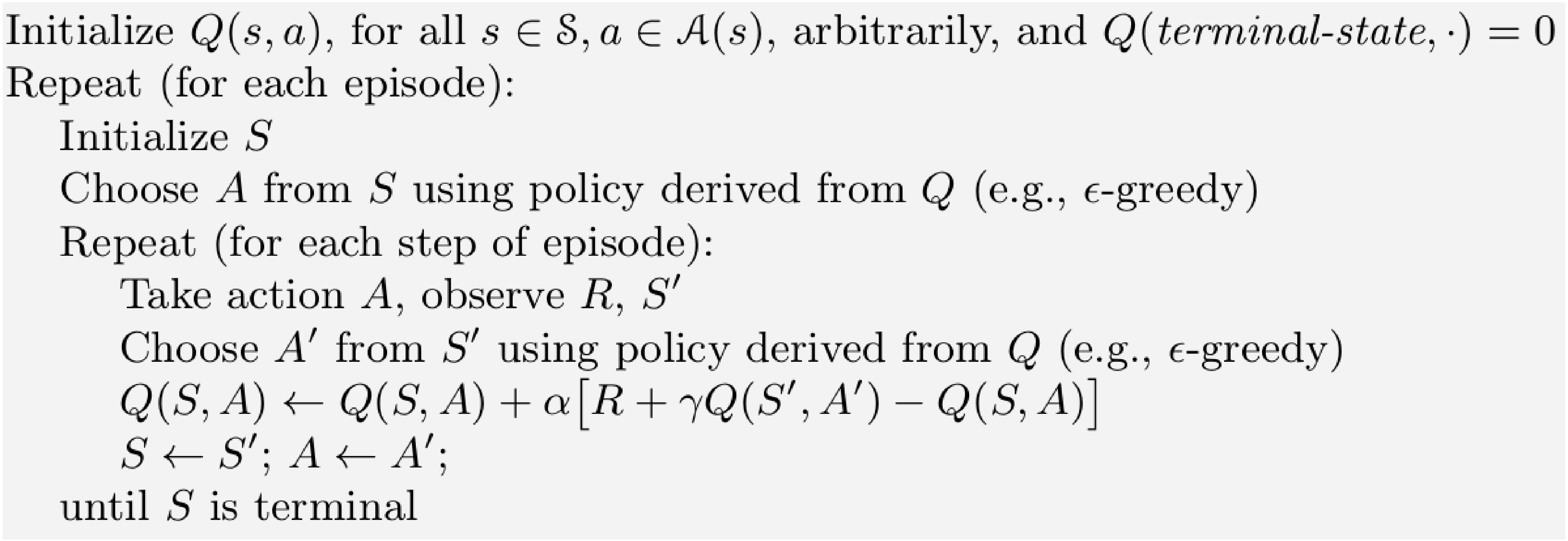
Update rule for Q-Learning:

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

This is the only difference between
SARSA and Q-Learning

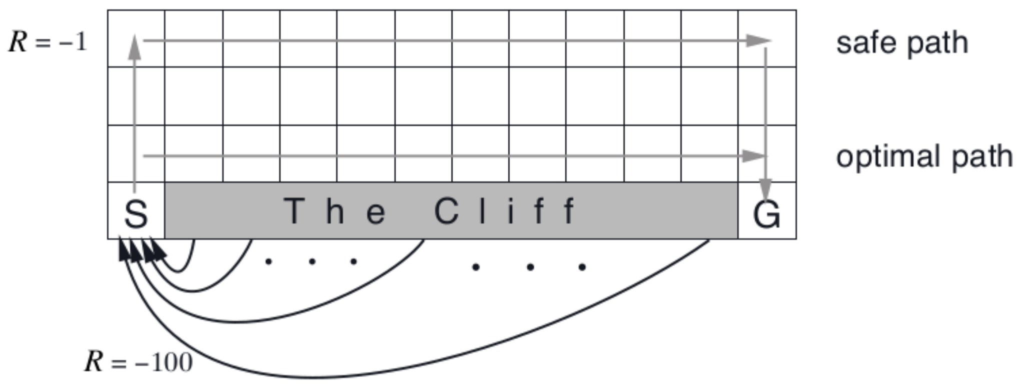
Q-Learning: a variation on SARSA

Q-Learning:

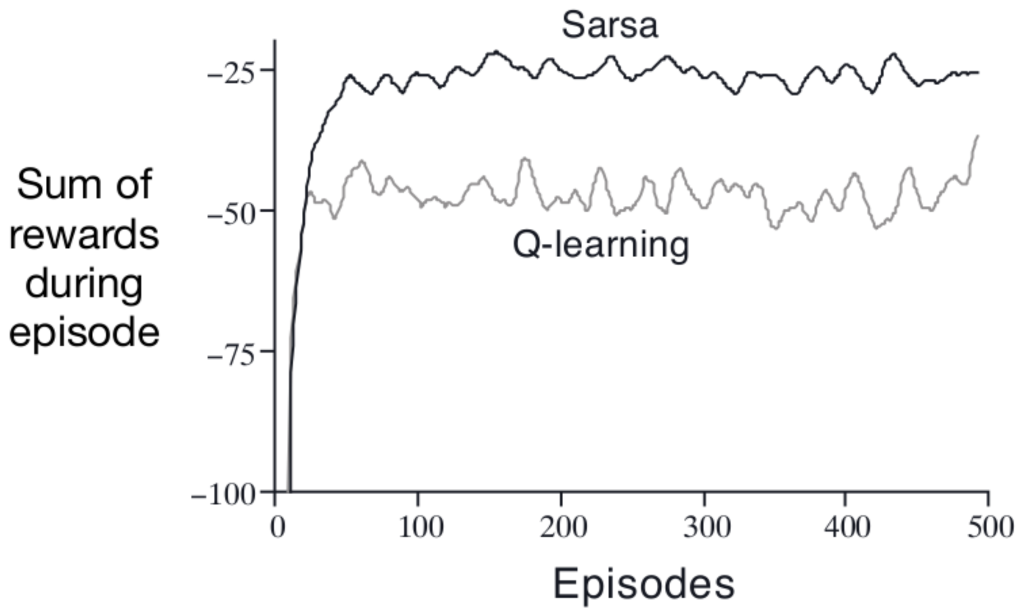


```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
  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

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?

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

In what sense are each of these solutions optimal?

Expected SARSA

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

Expected SARSA

Expected value of next state/action pair

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

Expected SARSA

Expected value of next state/action pair

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

Compare this w/ standard SARSA:

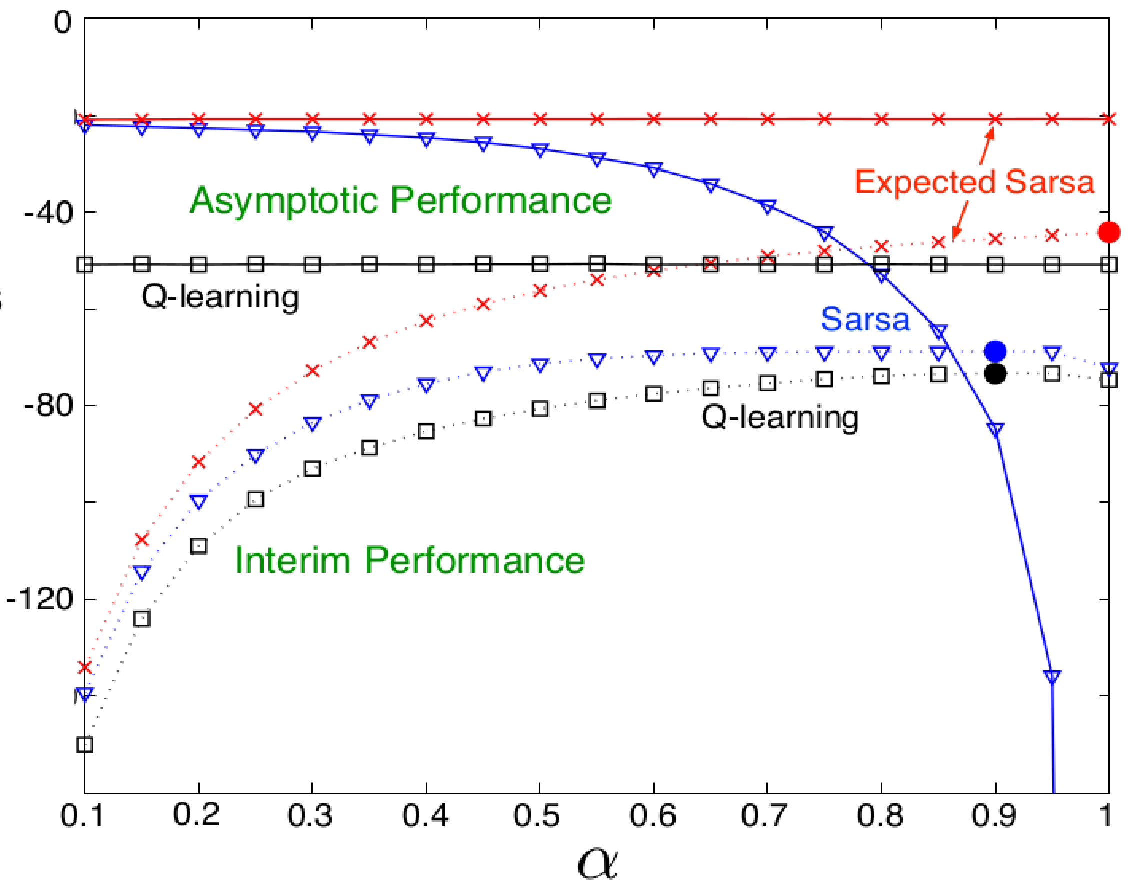
$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)]$$

Expected SARSA

Expected value of next state/action pair

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

Sum of rewards per episode

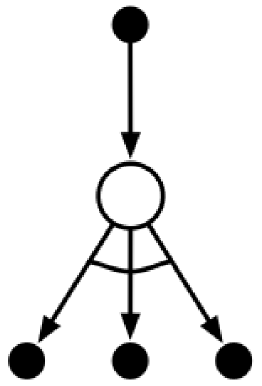


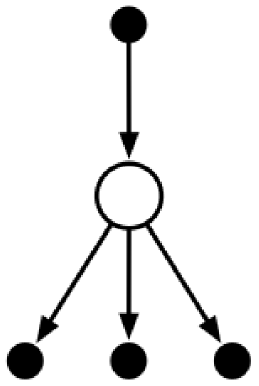
Interim performance: after first 100 episodes

Asymptotic performance: after first 100k episodes

Details:
– cliff walking task, eps=0.1

Backup diagrams

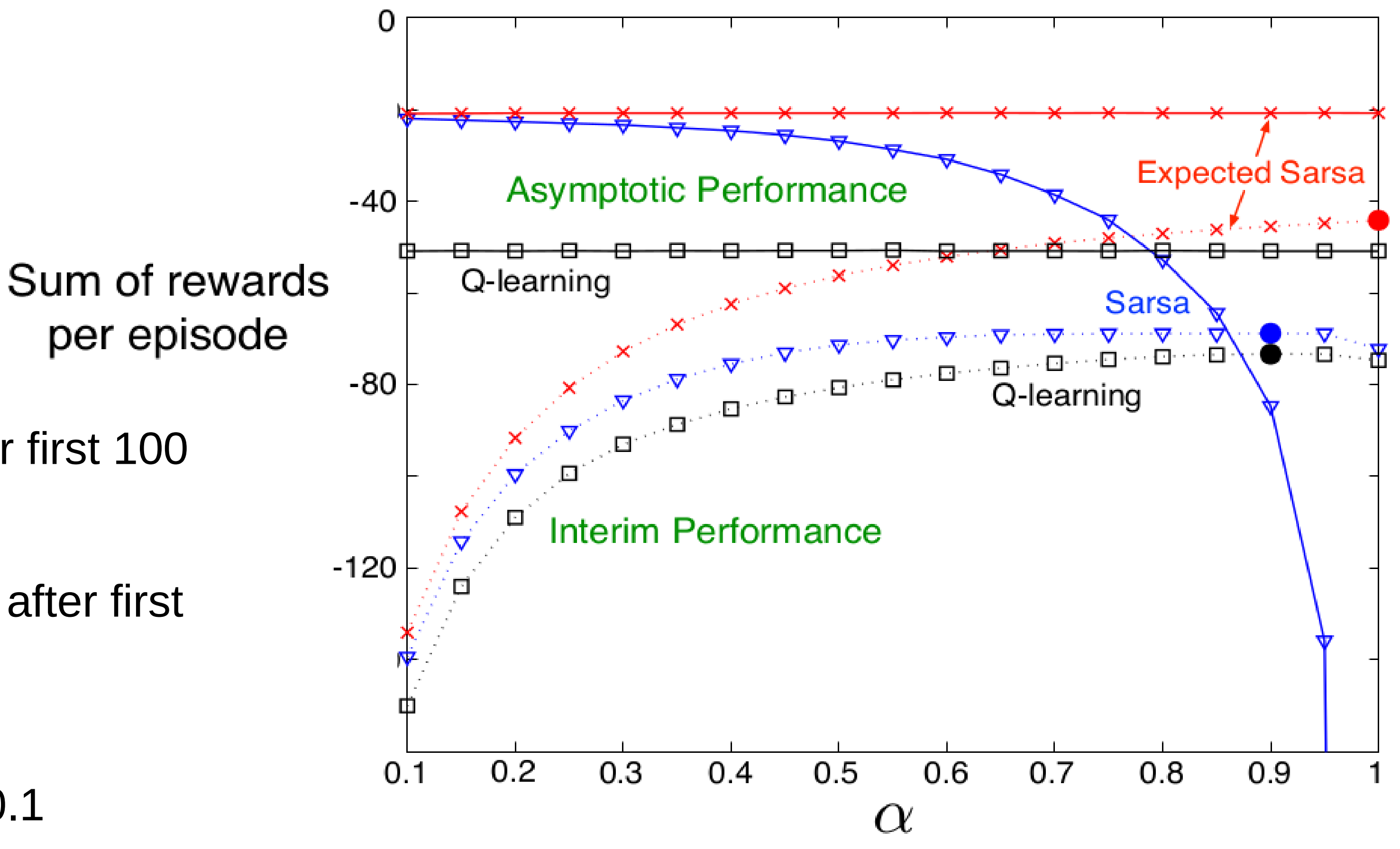

Q-learning


Expected Sarsa

Think-pair-share

Why does SARSA perf drop off for larger alpha values? Why exp-SARSA not drop off?

Under what conditions would off-policy exp-SARSA and Q-learning be equivalent?



Interim performance: after first 100 episodes

Asymptotic performance: after first 100k episodes

Details:
– cliff walking task, eps=0.1

Maximization bias in Q-learning

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

The problem:

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

Maximization bias in Q-learning

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

The problem:

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

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

Maximization bias in Q-learning

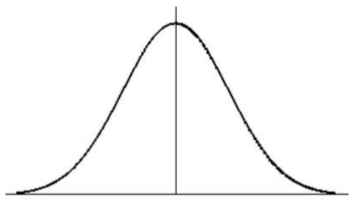
Maximization over random samples is not a good estimate of the max of the expected values

The problem:

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

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

A “Gaussian” is the probability distribution corresponding to the “bell curve”:



Maximization bias in Q-learning

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

<u>The problem:</u>

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

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 #1:
– estimate sample mean of each variable, $\hat{a}$ and $\hat{b}$
– then, calculate $\max(\hat{a}, \hat{b})$

Think-pair-share

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

The problem:

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

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

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

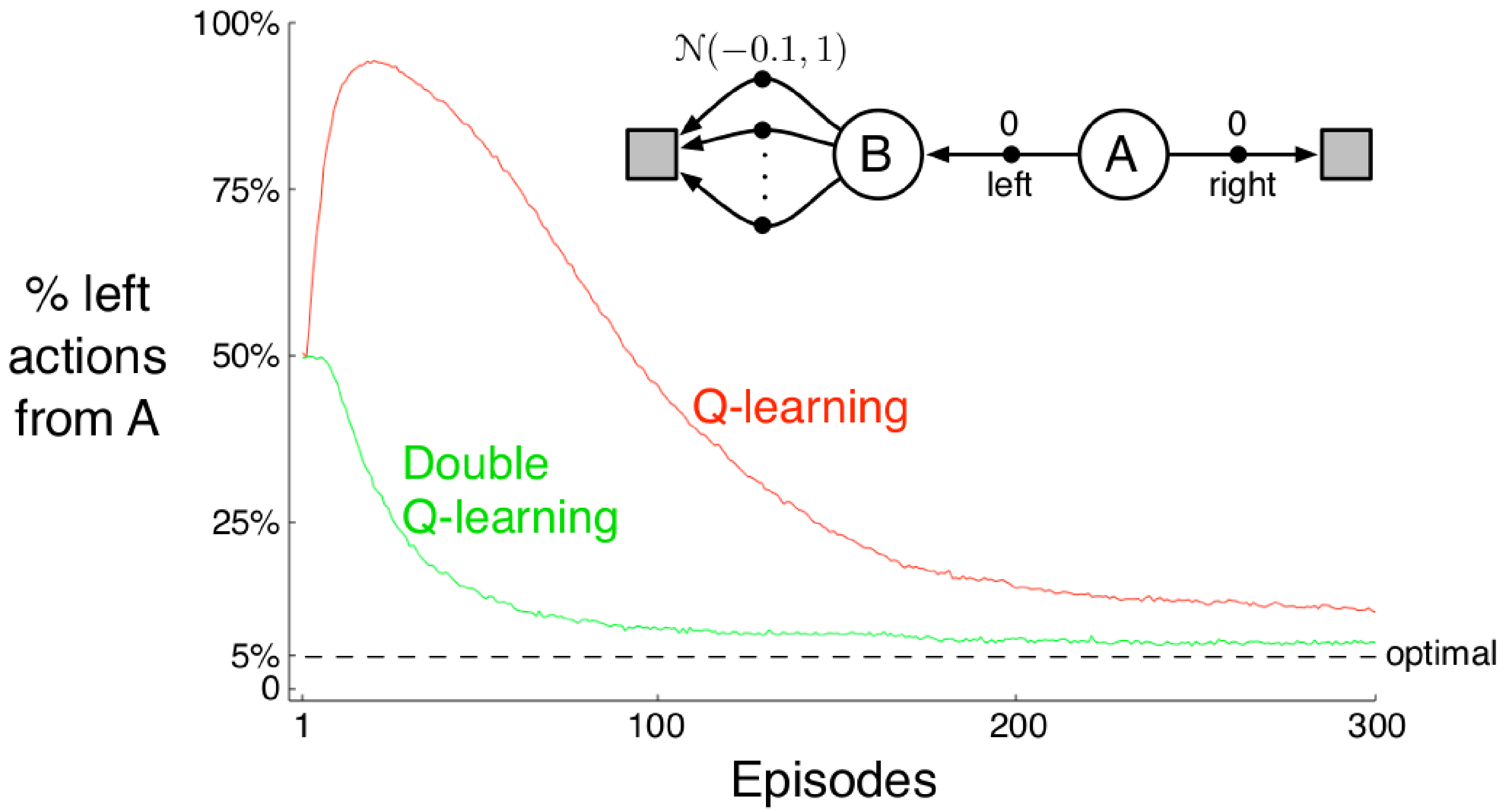
Solution #1:
– estimate sample mean of each variable, $\hat{a}$ and $\hat{b}$
– then, calculate $\max(\hat{a}, \hat{b})$

Questions:

1. For 2 Gaussian rand variables, how often does this occur: $\max(\hat{a}, \hat{b}) > \max(\mathbb{E}(a), \mathbb{E}(b))$

2. Does the problem get worse or better w/ more variables (i.e. more actions)?

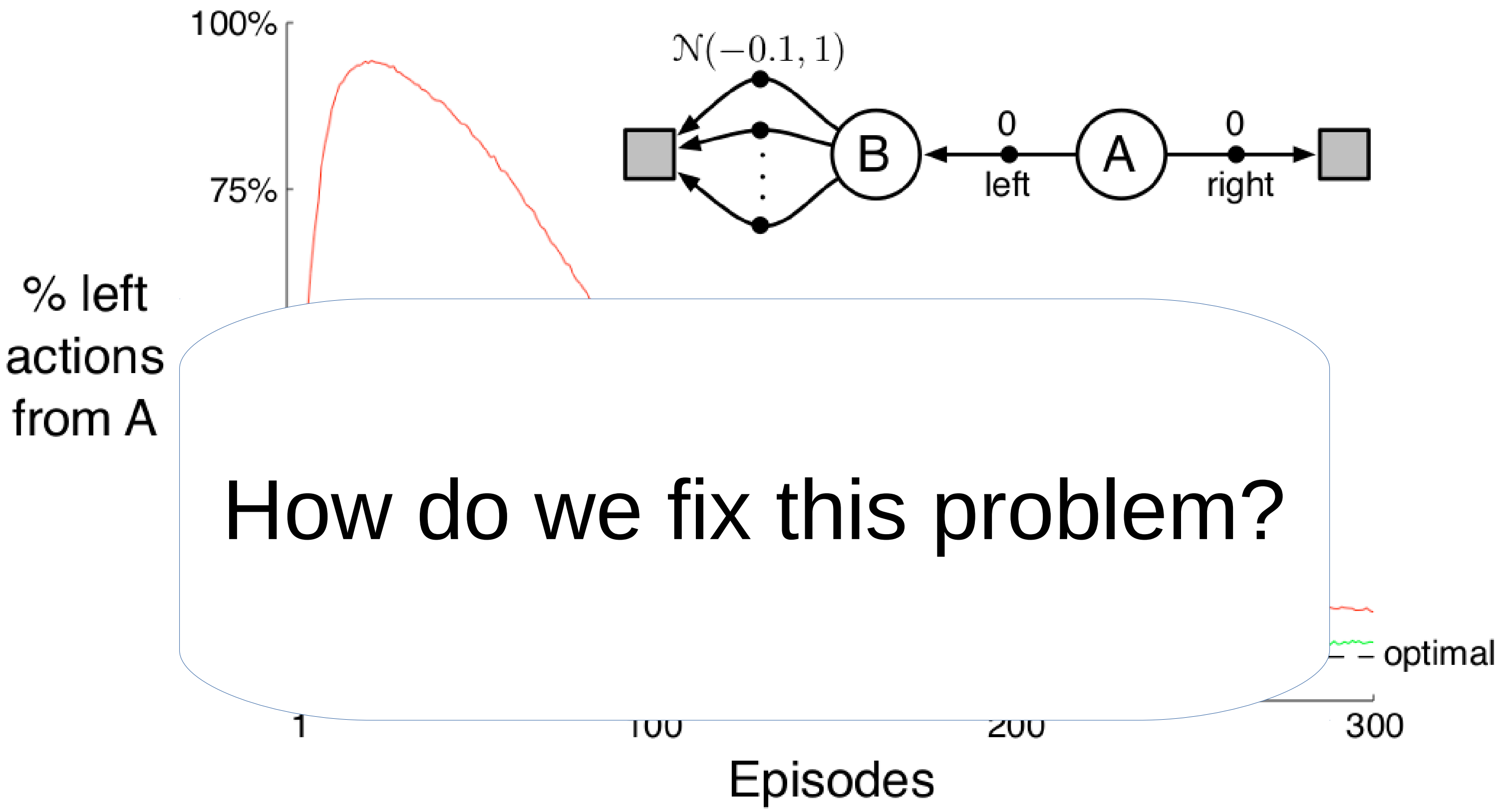
Why maximization bias is a problem



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

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

Why maximization bias is a problem



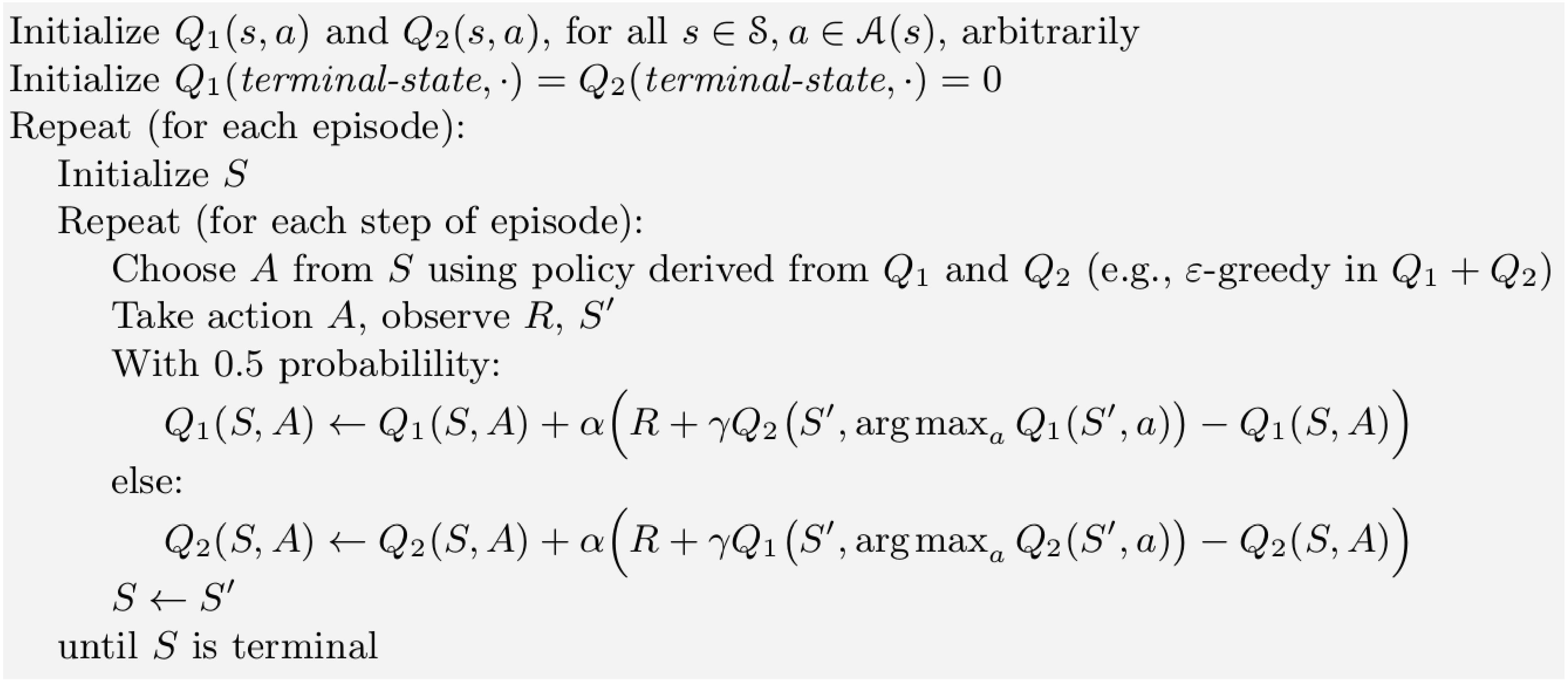
How do we fix this problem?

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

Two states, two actions
Rewards:
– going right from A always leads to zero reward and then terminates
– going left from B leads to 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., $\varepsilon$-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., $\varepsilon$-greedy in $Q_1 + Q_2$)
        Take action $A$, observe $R$, $S'$
        With 0.5 probabilility:
            $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma \boxed{Q_2\big(S', \arg\max_a Q_1(S',a)\big)} - Q_1(S,A)\Big)$
        else:
            $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma \boxed{Q_1\big(S', \arg\max_a Q_2(S',a)\big)} - Q_2(S,A)\Big)$
        $S \leftarrow S'$
    until $S$ is terminal
```

How exactly does this fix the problem?

Afterstate Representation

Sometimes, we know exactly how an action will effect the env't, but it is less clear how state will evolve after that.

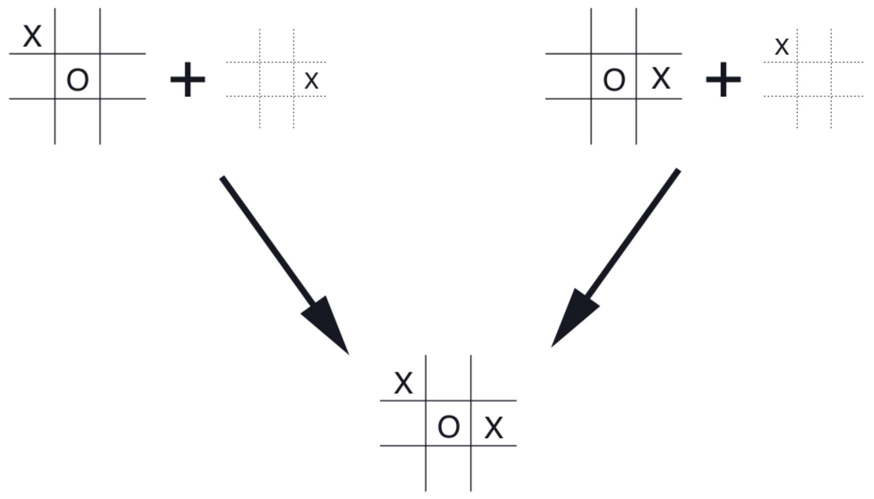
– afterstates can help in this situation

Idea: reason in terms of the state of the world *after* executing the action currently under consideration.

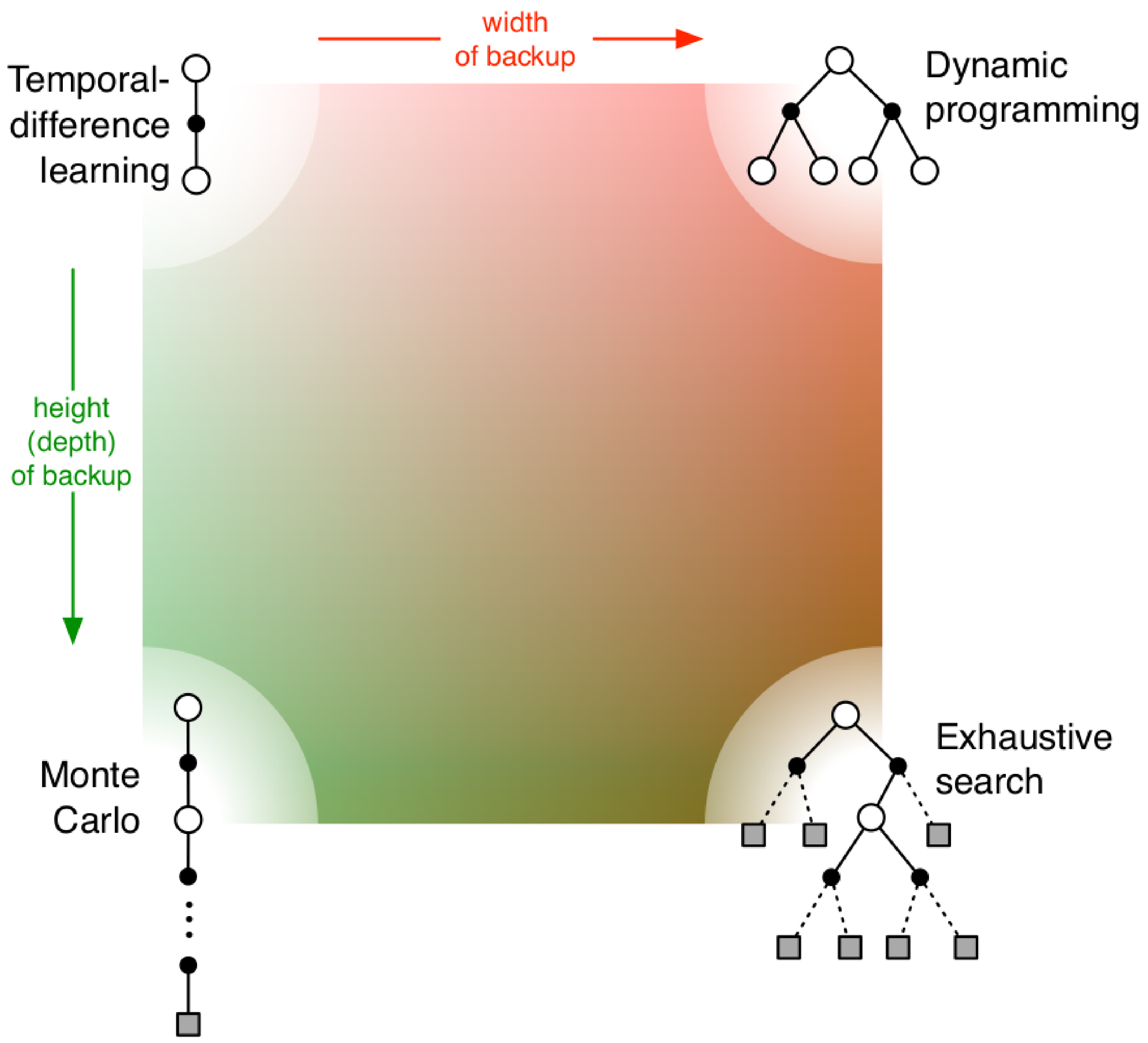
– estimate Q-Table in terms of *s'* rather than *s,a*

– easier to estimate state-values vs action-values

Example: tic-tac-toe



Unified view


width
of backup
Temporal-
difference
learning
Dynamic
programming
height
(depth)
of backup
Monte
Carlo
Exhaustive
search