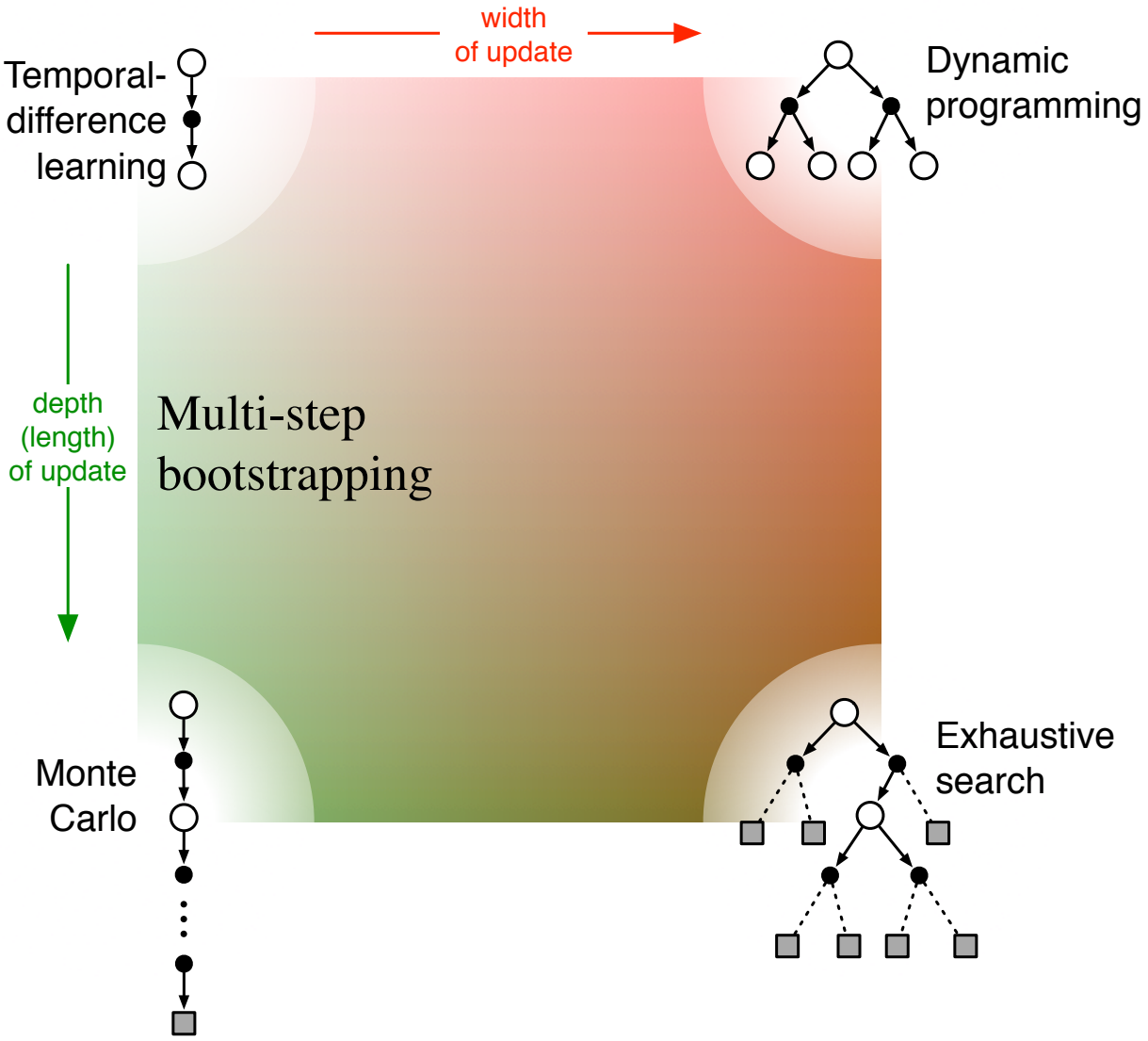
Temporal Difference Learning

- Introduce Temporal Difference (TD) learning
- Focus first on policy evaluation, or prediction, methods
- Compare efficiency of TD learning with MC learning
- Then extend to control methods

Unified View



TD methods bootstrap and sample

- Bootstrapping: update involves an estimate of the value function
  - TD and DP methods bootstrap
  - MC methods **do not** bootstrap
- Sampling: update **does not** involve an expected value
  - TD and MC method sample
  - Classical DP **does not** sample

Prediction vs Control

- For the control problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of **generalized policy iteration (GPI)**.
- The differences in the methods are primarily differences in their approaches to the prediction problem.

Prediction vs Control

- **Dynamic Programming:**
  - Explore the given dynamics model to learn a value function
  - Iterate until V or Q or π converge using explicit P(s'|s,a)

Prediction vs Control

- **Monte Carlo and Temporal Differences:**
  - Learn from experience following a policy
  - Update estimate V of v_π for the nonterminal states S_t that were seen in the sampled episode

- **Monte Carlo:**
  - Sample from P(s'|s,a) *until goal state*, then update backwards

- **Temporal Differences**
  - Sample from P(s'|s,a) *until next state*, then update immediately

Recall: Multi-armed Bandits

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

Recall: Multi-armed Bandits

Averaging $\longrightarrow$ learning rule

- To simplify notation, let us focus on one action
  - We consider only its rewards, and its estimate after $n+1$ rewards:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- How can we do this incrementally (without storing all the rewards)?
- Could store a running sum and count (and divide), or equivalently:

$$Q_{n+1} = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

- This is a standard form for learning/update rules:

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \; \Big[Target - OldEstimate\Big]$$

TD Prediction

Policy Evaluation (the prediction problem):
for a given policy π, compute the state-value function v_π

Consider a very simple every-visit Monte Carlo method
suitable for non-stationary environments:

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

Target: the actual return $G_t = R_1 + R_2 + \ldots + R_{t-1}$ after time t

Step-size: α is a constant step-size

TD Prediction

Monte Carlo Method: uses a simple backup suitable for non-stationary environments:

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

Target: the *actual* return $G_t = R_1 + R_2 + \ldots + R_{t-1}$ after time t

Step-size: α is a constant step-size

TD Method: The simplest temporal-difference method **TD(0)**:

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

target: an *estimate* of the return

TD target for prediction

- The TD target: $R_{t+1} + \gamma v_\pi(S_{t+1})$
  - it is an *estimate* like MC target because it **samples** the expected value
  - it is an *estimate* like the DP target because it uses the current estimate of V instead of v_π

Tabular 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

Agent program

Environment program

Experiment program

Tabular 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

Agent program

Environment program

Experiment program

Tabular 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

Agent program

Environment program

Experiment program

Tabular 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

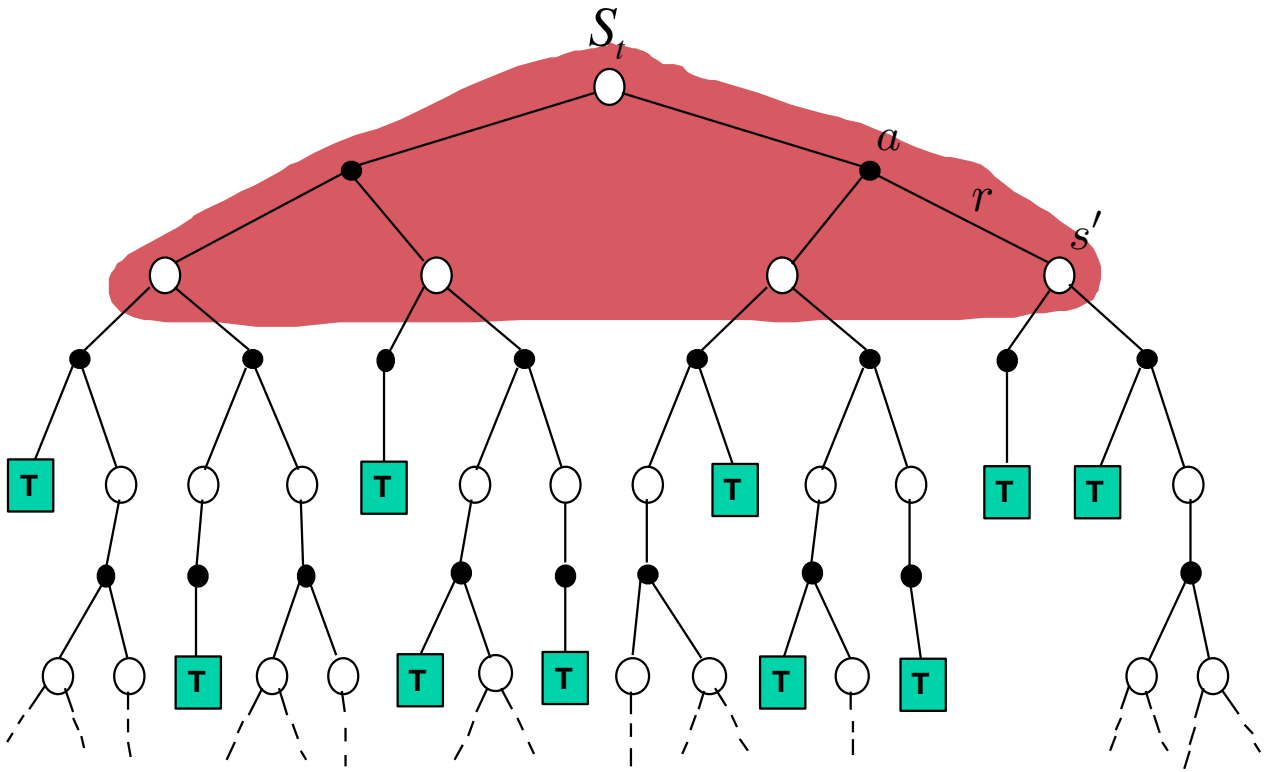
Agent program

Environment program

Experiment program

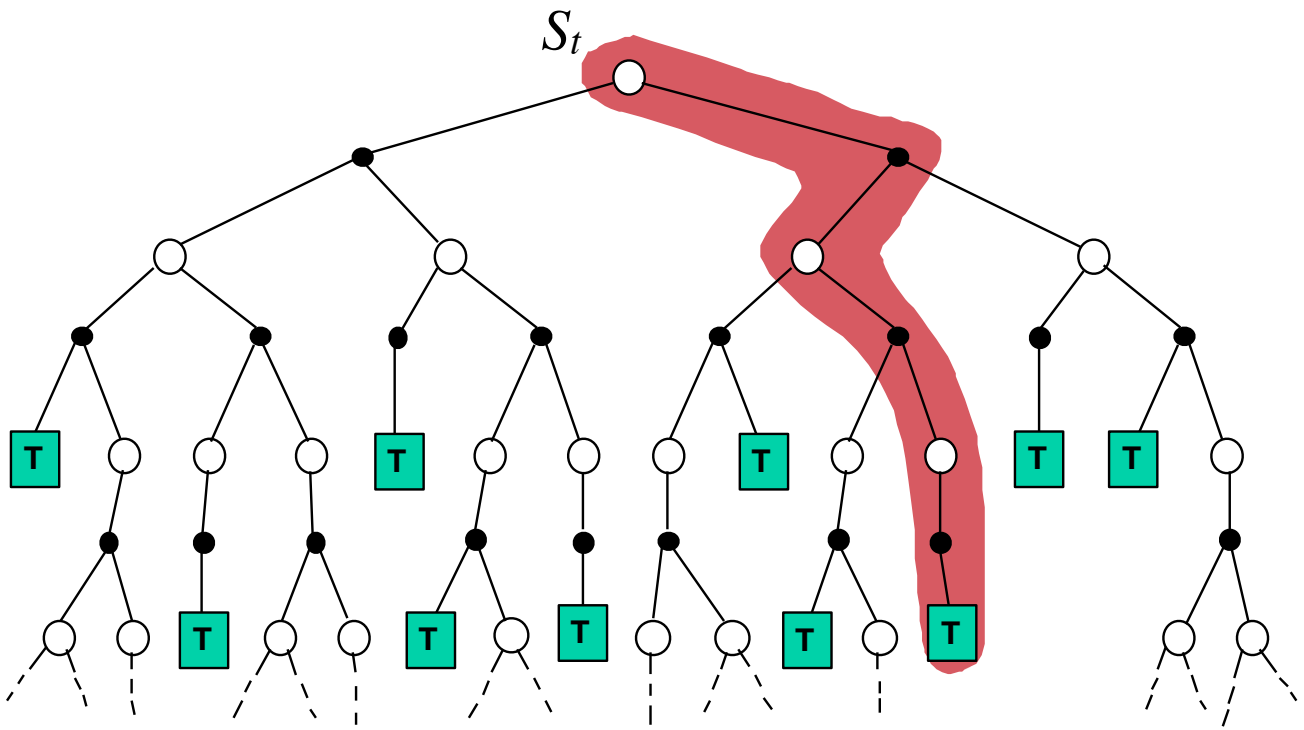
Dynamic programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')]$$



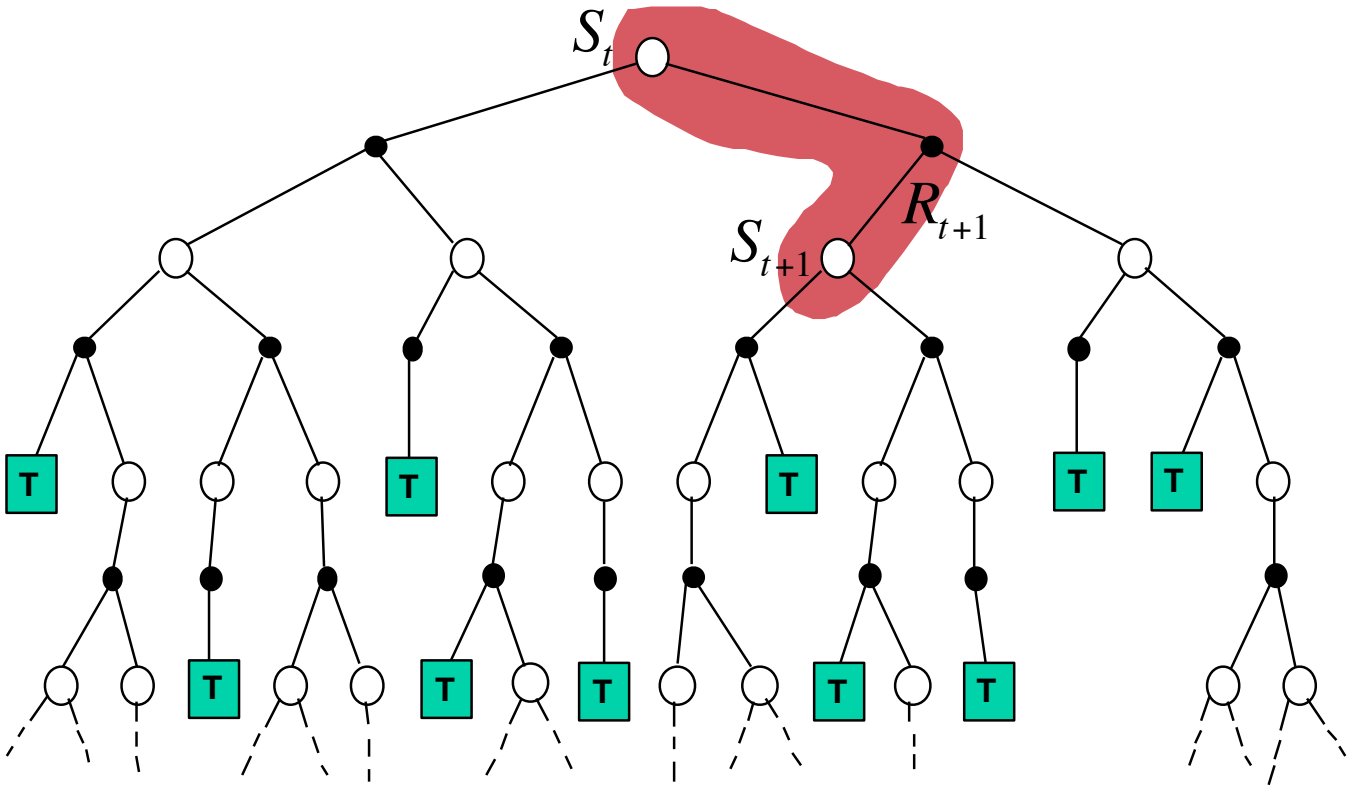
Simple Monte Carlo

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



Simplest TD method

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



Example: Driving Home

- Consider driving home:
  - each day you drive home
  - your goal is to try and predict how long it will take at particular stages
  - when you leave office you note the time, day, & other relevant info
- Consider the policy evaluation or prediction task

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

Driving home as an RL problem

- Rewards = 1 per step (if we were minimizing travel time what would reward be?)
- $\gamma = 1$
- G_t = time to go from state S_t
- $V(S_t)$ = expected time to get home from S_t

Updating our predictions

- Goal: update the prediction of total time leaving from office, while driving home
- With MC we would need to wait for a termination—until we get home—then calculate G_t for each step of episode, then apply our updates

Driving home

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

Driving home

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

- Task: update the value function as we go, based on observed elapsed time—Reward column

Driving home

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

Driving home

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

- update V(office) with $\alpha = 1$?

Driving home

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

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$

Driving home

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

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$

Driving home

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

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$

Driving home

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

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?

Driving home

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

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?
  - V(car) = 30; $\Delta = -5$

Driving home

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

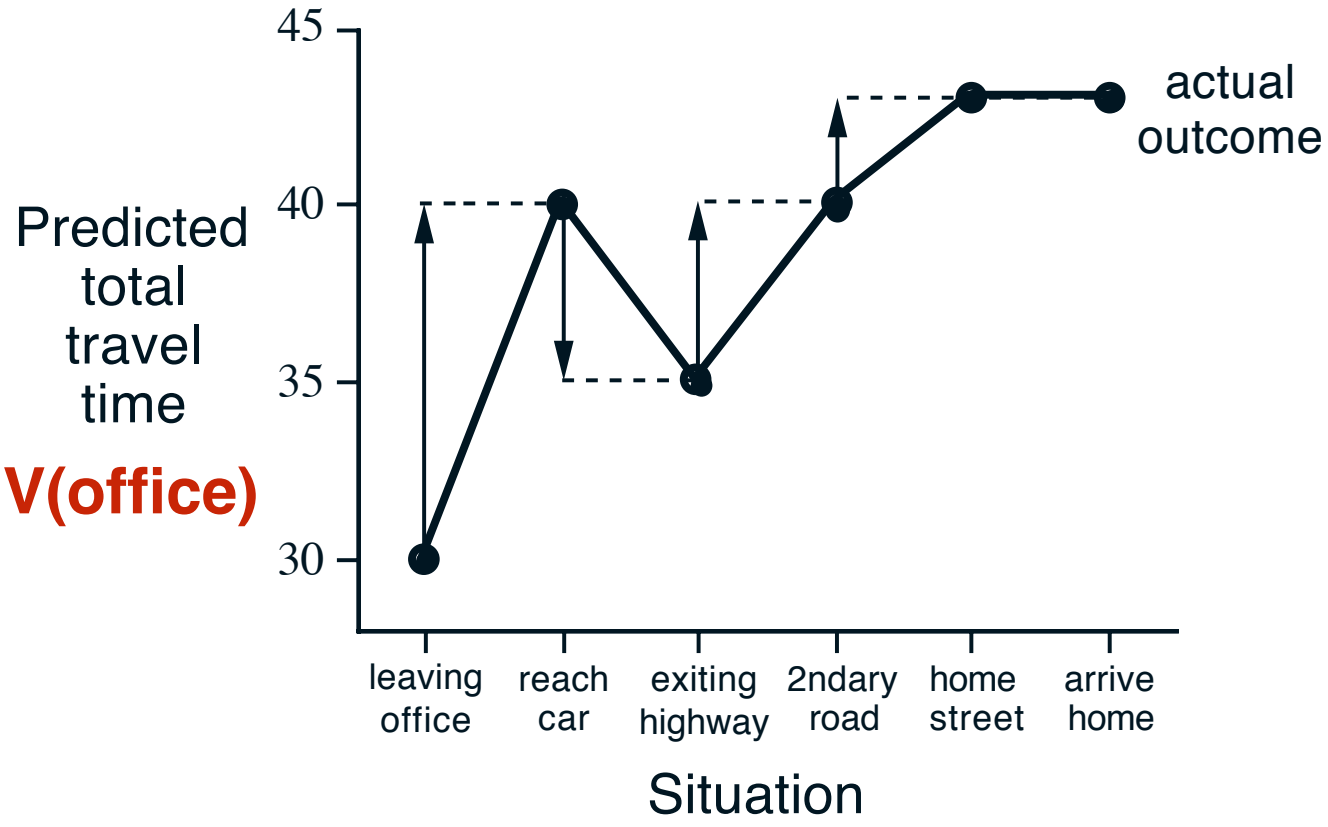
- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?
  - V(car) = 30; $\Delta = -5$
- update V(exit)?

Driving home

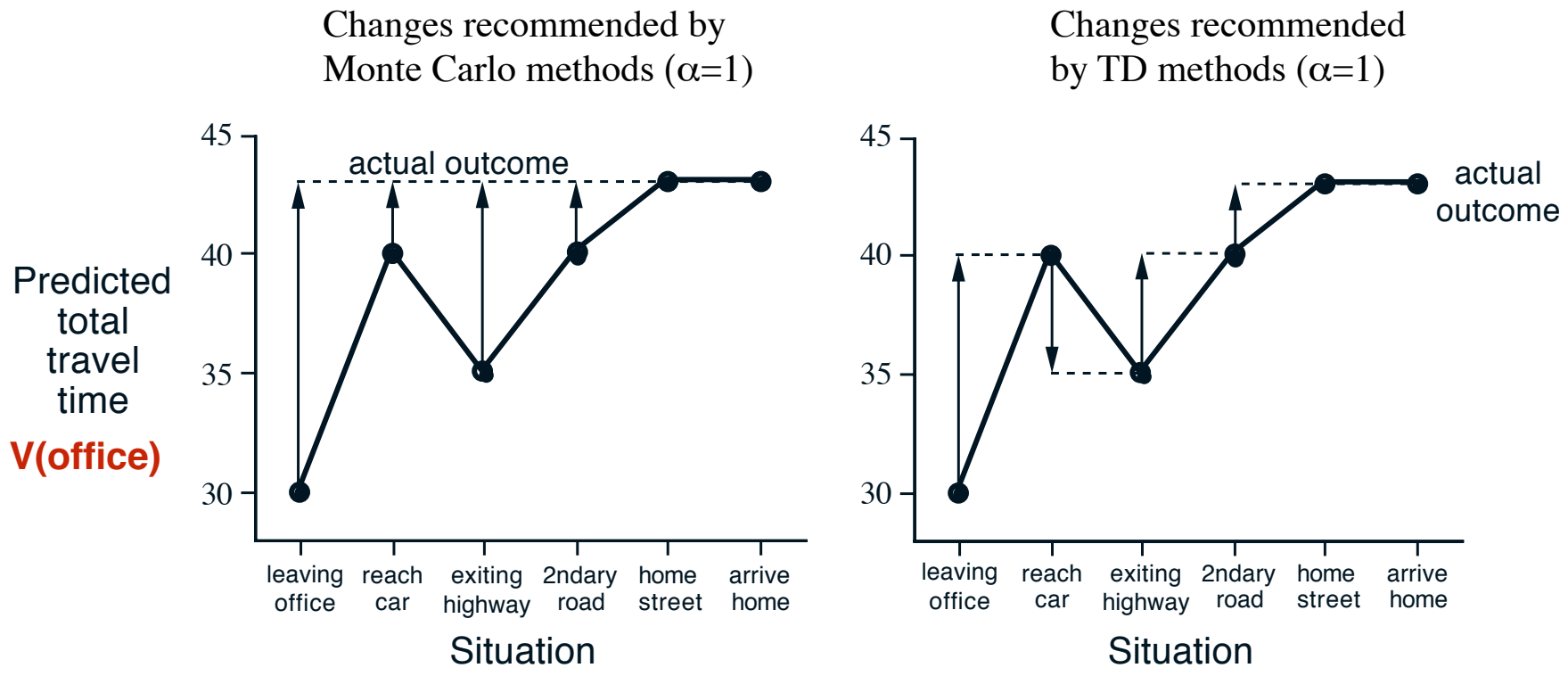
State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

- update V(office) with $\alpha = 1$?
  - $V(s) = V(s) + \alpha[R_{t+1} + \gamma V(s') - V(s)]$
  - $V(office) = V(office) + \alpha[R_{t+1} + \gamma V(car) - V(office)]$
  - new V(office) = 40; $\Delta = +10$
- update V(car)?
  - V(car) = 30; $\Delta = -5$
- update V(exit)?
  - V(exit) = 20; $\Delta = +5$

Changes recommended by TD methods ($\alpha = 1$)



Driving Home



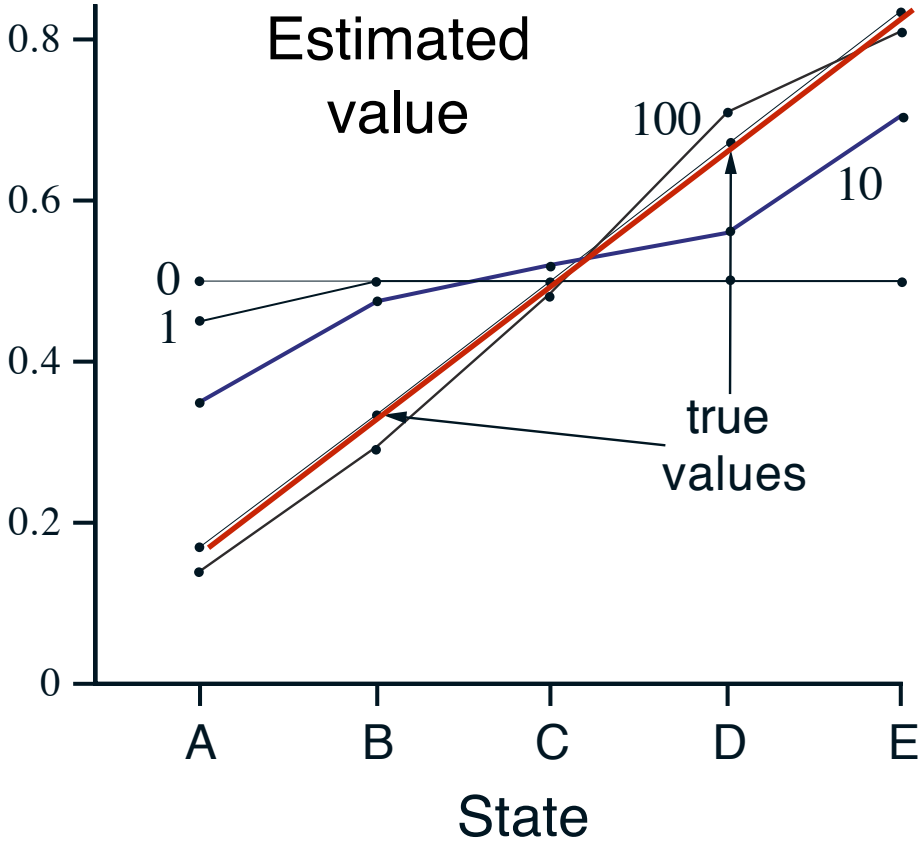
Advantages of TD learning

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

Random walk

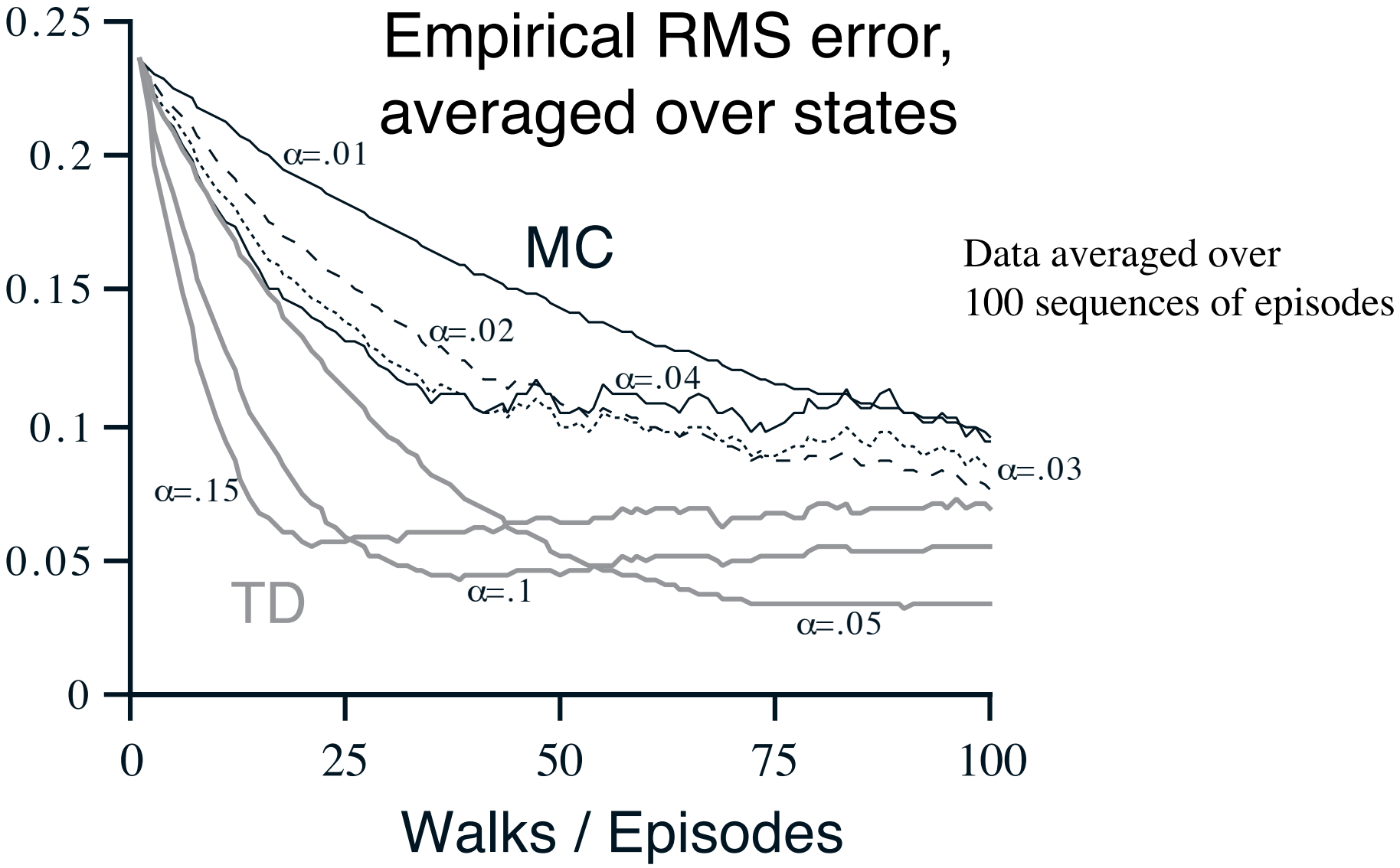
- C is start state, episodic, undiscounted γ = 1
- π is left or right with equal probability in all states
- termination at either end
- rewards +1 on **right** termination, 0 otherwise
- what does $v_\pi(s)$ tell us?
  - probability of termination on right side from each state, under random policy
  - what is v_π =[A B C D E]?
    - v_π = [1/6 2/6 3/6 4/6 5/6]
- Initialize V(s) = 0.5 $\forall\ s\ \in\ \mathcal{S}$

Values learned by TD from one run, after various numbers of episodes



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

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Batch Updating in TD and MC methods

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

Compute updates according to TD or MC, but only update estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating, TD converges for sufficiently small α.

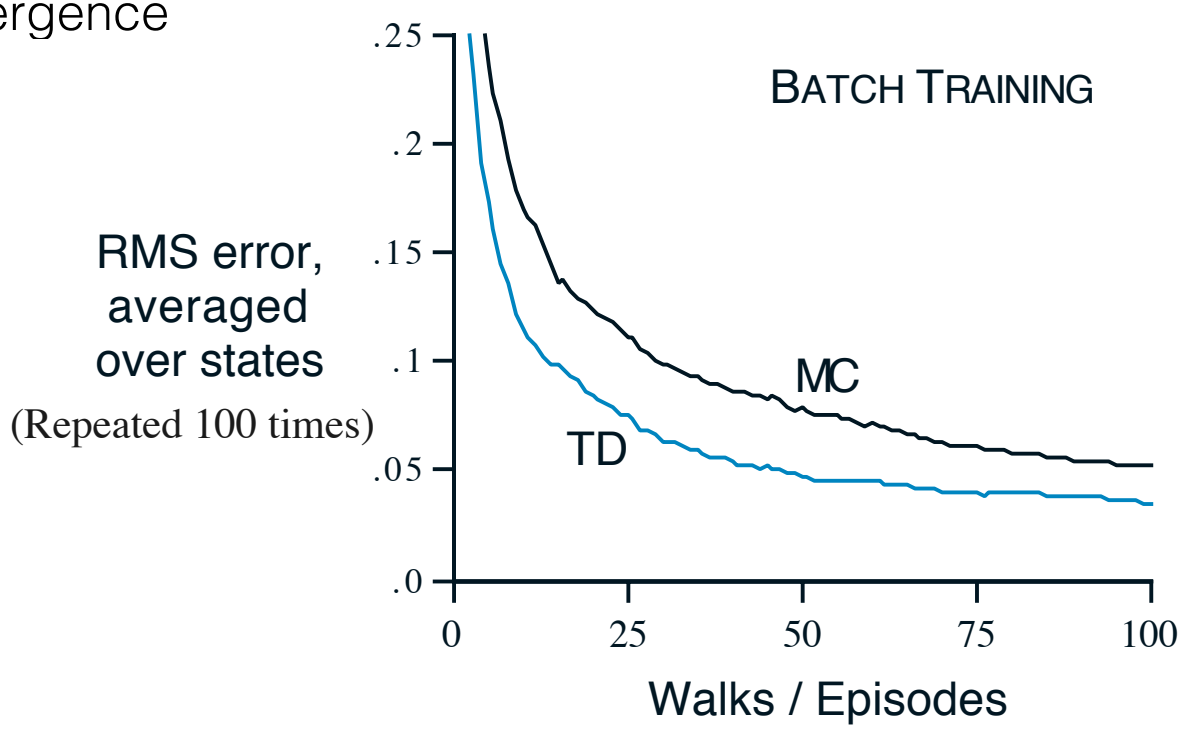
Constant-α MC also converges under these conditions, but to a difference answer!

Random Walk under Batch Updating

- After each new episode, all episodes seen so far are treated as a batch
- This growing batch is repeatedly processed by TD and MC until convergence

Random Walk under Batch Updating

- After each new episode, all episodes seen so far are treated as a batch
- This growing batch is repeatedly processed by TD and MC until convergence



Optimality of 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

Advantages of TD

Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data

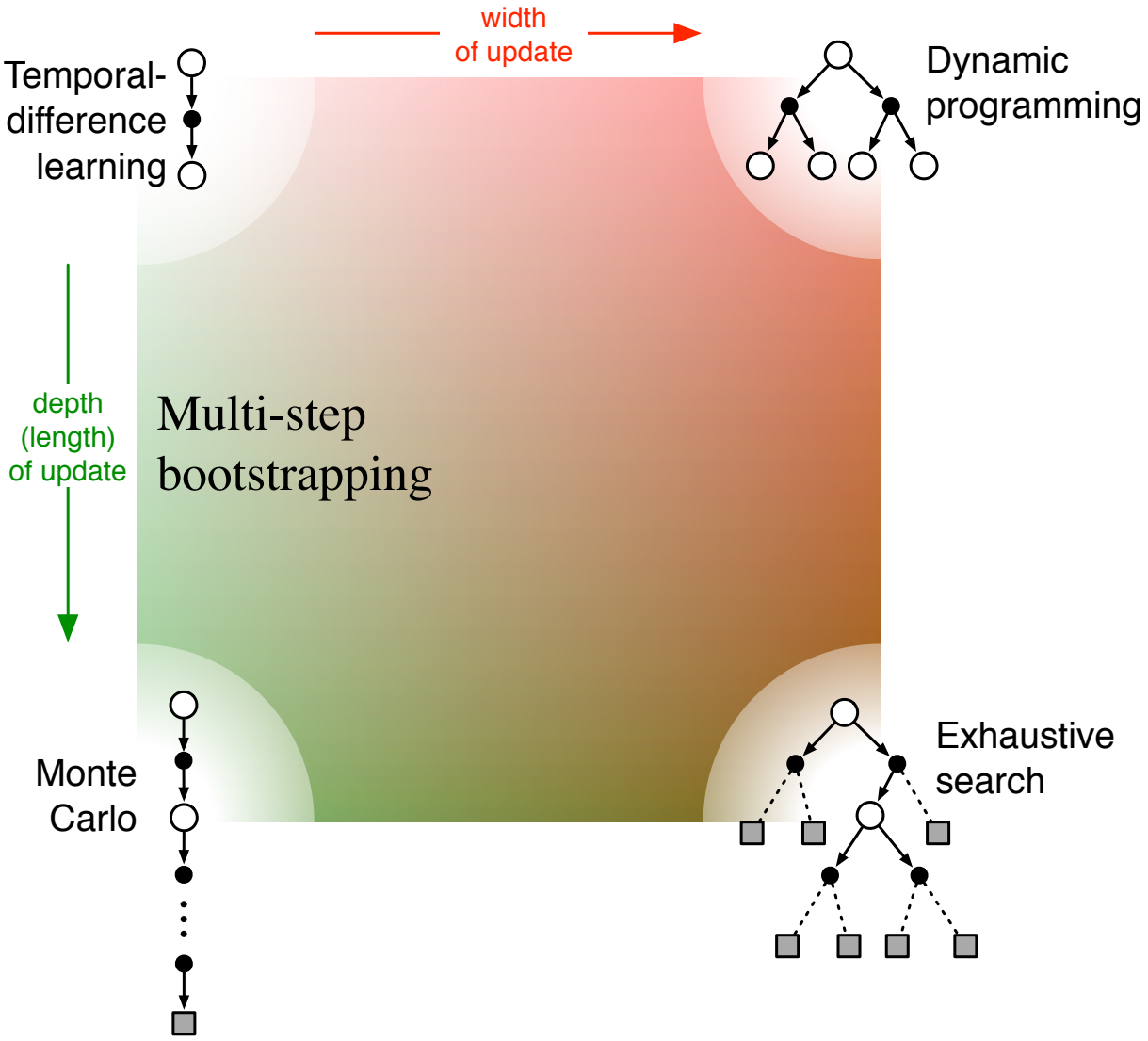
Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data
- This helps explain why TD methods converge more quickly than MC in the batch setting

Advantages of TD

- If the process is Markov, then we expect the TD estimate to produce lower error on future data
- This helps explain why TD methods converge more quickly than MC in the batch setting
- TD(0) makes progress towards the certainty-equivalence estimate without explicitly building the model!

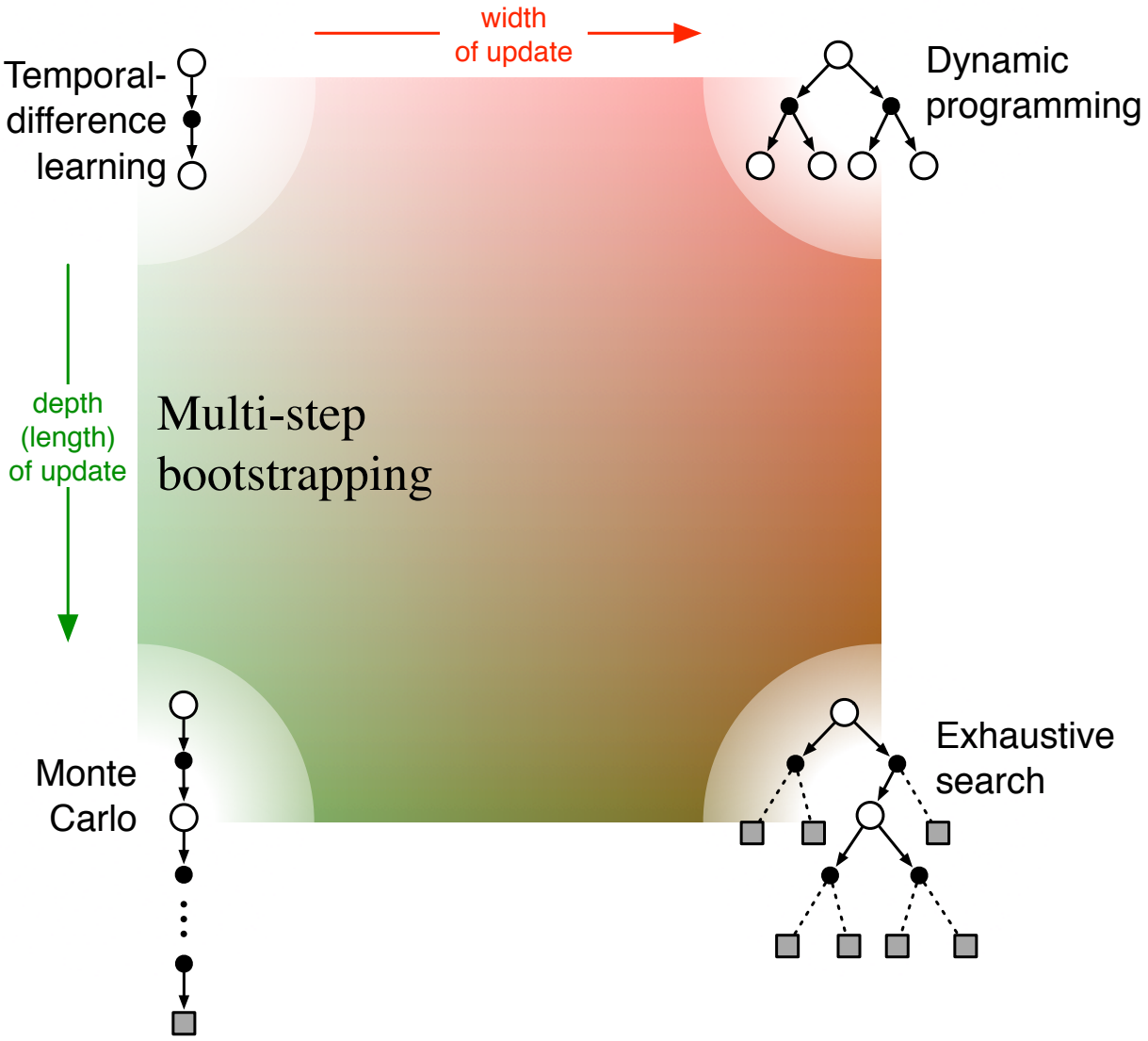
Unified View



Summary so far

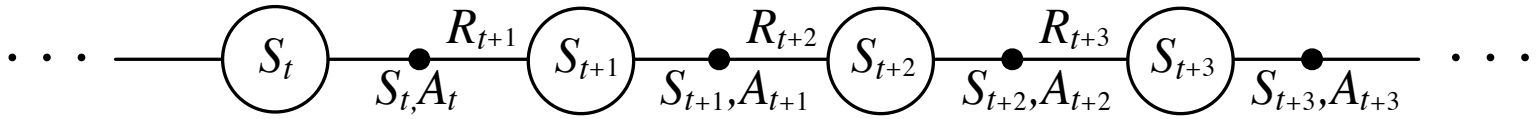
- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data

Unified View



Learning An Action-Value Function

Estimate q_π for the current policy π



After every transition from a nonterminal state, S_t, do this:

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

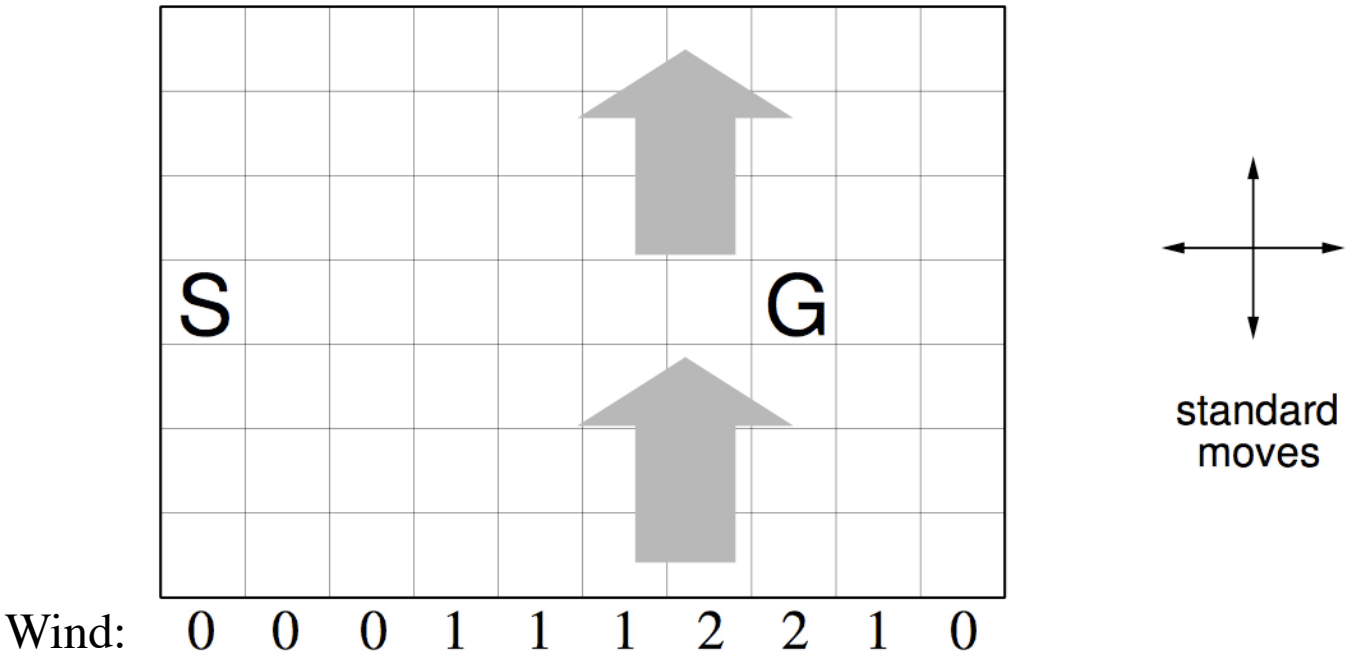
If S_{t+1} is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

Sarsa: On-Policy TD Control

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

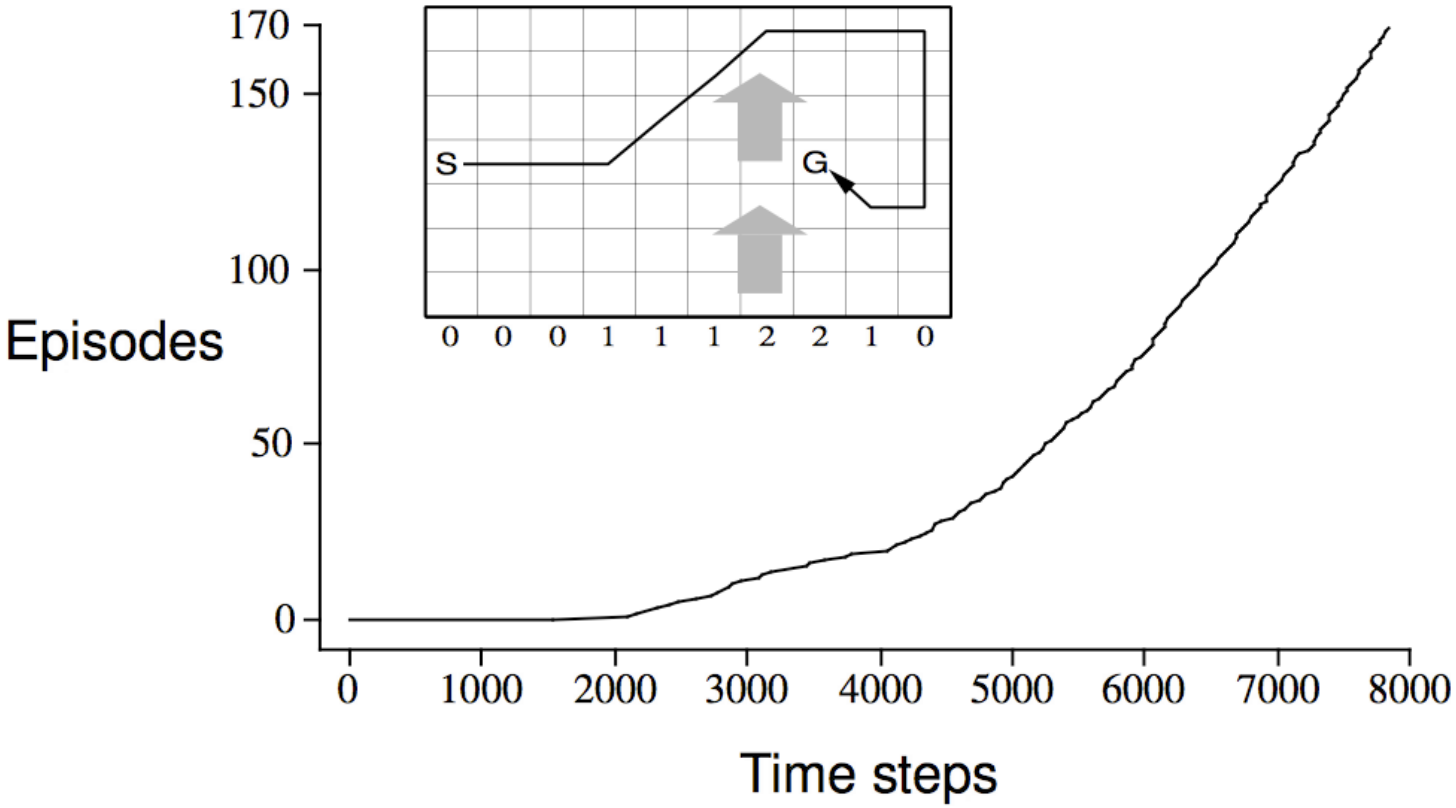
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Windy Gridworld



undiscounted, episodic, reward = –1 until goal

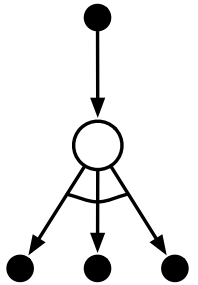
Results of Sarsa on the Windy Gridworld



Q-Learning: Off-Policy TD Control

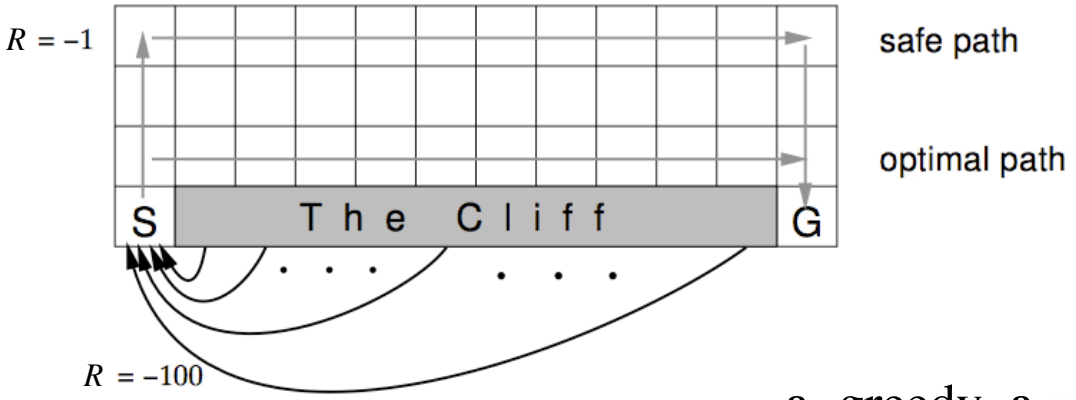
One-step Q-learning:

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

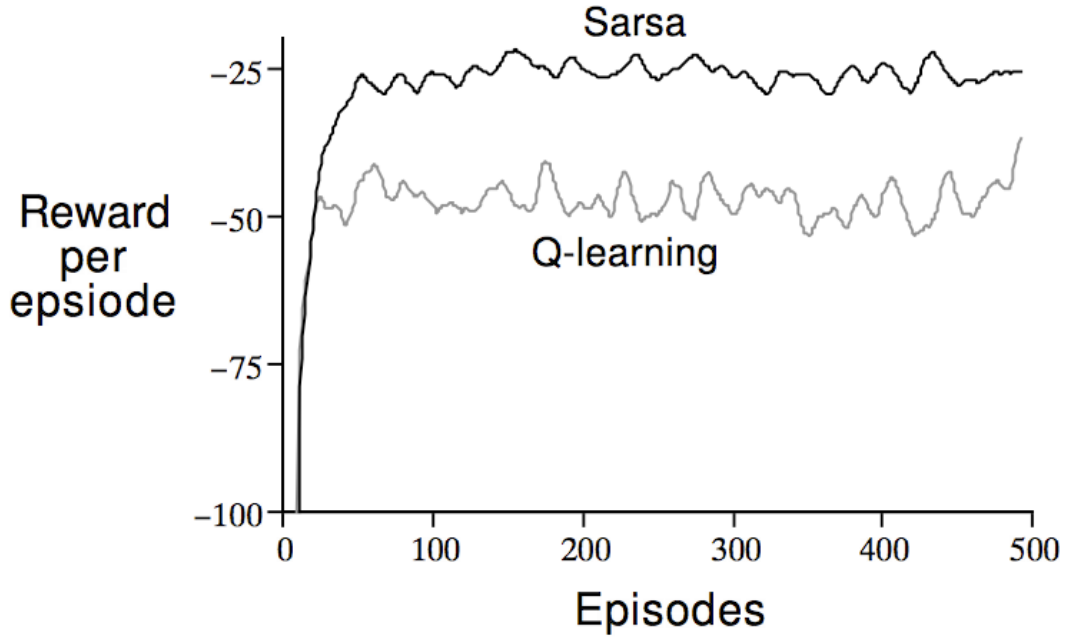


Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$;
    until S is terminal

Cliffwalking



ε−greedy, ε = 0.1



SARSA vs. Q-Learning

- They are very similar!
- **SARSA**
  - on-policy
  - Chooses an action, not necessarily the best one, sees the result
  - Then updates it's value function with that knowledge
  - Will converge eventually, but more slowly than Learning
- **Q-Learning**
  - Off-policy
  - Chooses an action, sees the result
  - Then updates it's value function with a different action
  - Chooses the best action according to value function
  - **No Guarantee it will ever converge**
  - **In practice it's often faster and better than SARSA**

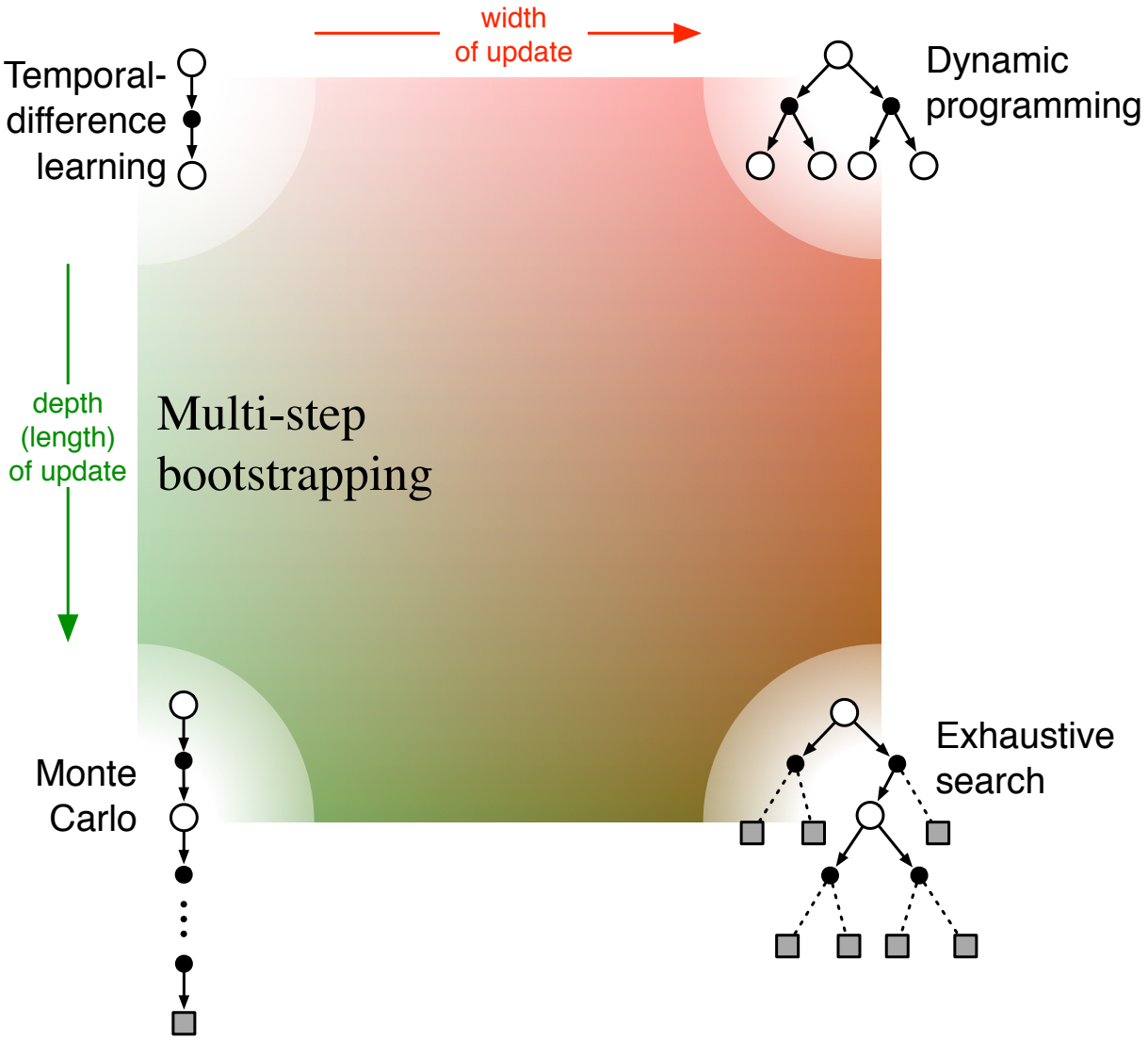
SARSA

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

QLearning

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$;
  until S is terminal

Unified View



Modified Versions of Sarsa and Q-Learning

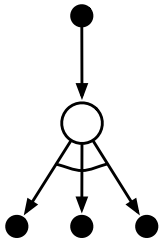
- The default versions of Sarsa and Q-Learning are vulnerable to some issues relating to *sample bias*
- Two basic ways to deal with this lead to new algorithms
  - **Expected Sarsa**
  - **Double Q-Learning**

Expected Sarsa

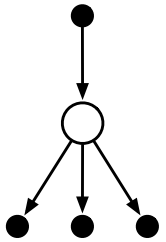
- Instead of the *sample* value-of-next-state, use the expectation!

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

How do we actually compute this?
For epsilon greedy
- $\pi(a \mid s) = 90\%$) for max a
- $\frac{1}{(|A| - 1)}$ for the other actions



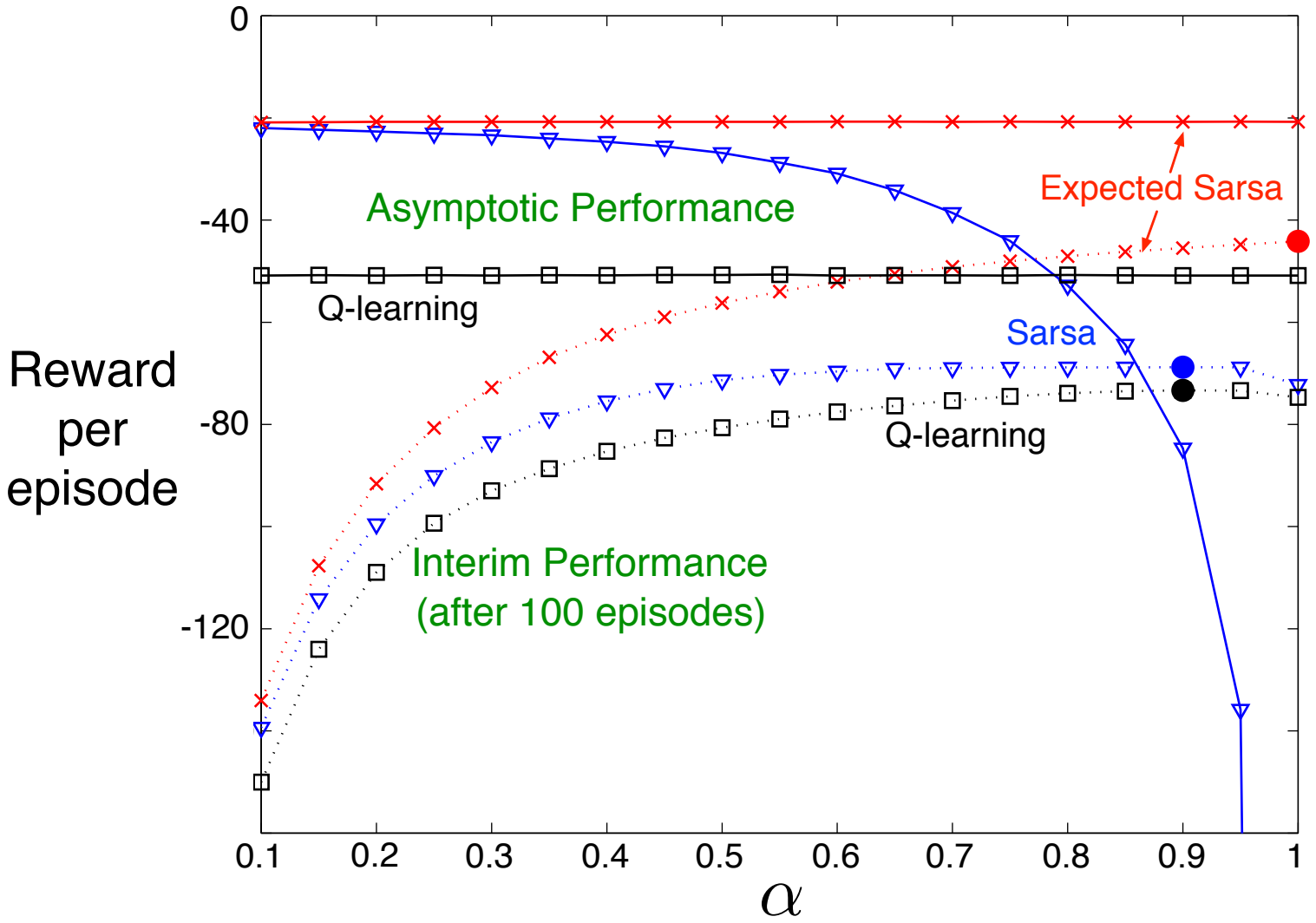
Q-learning



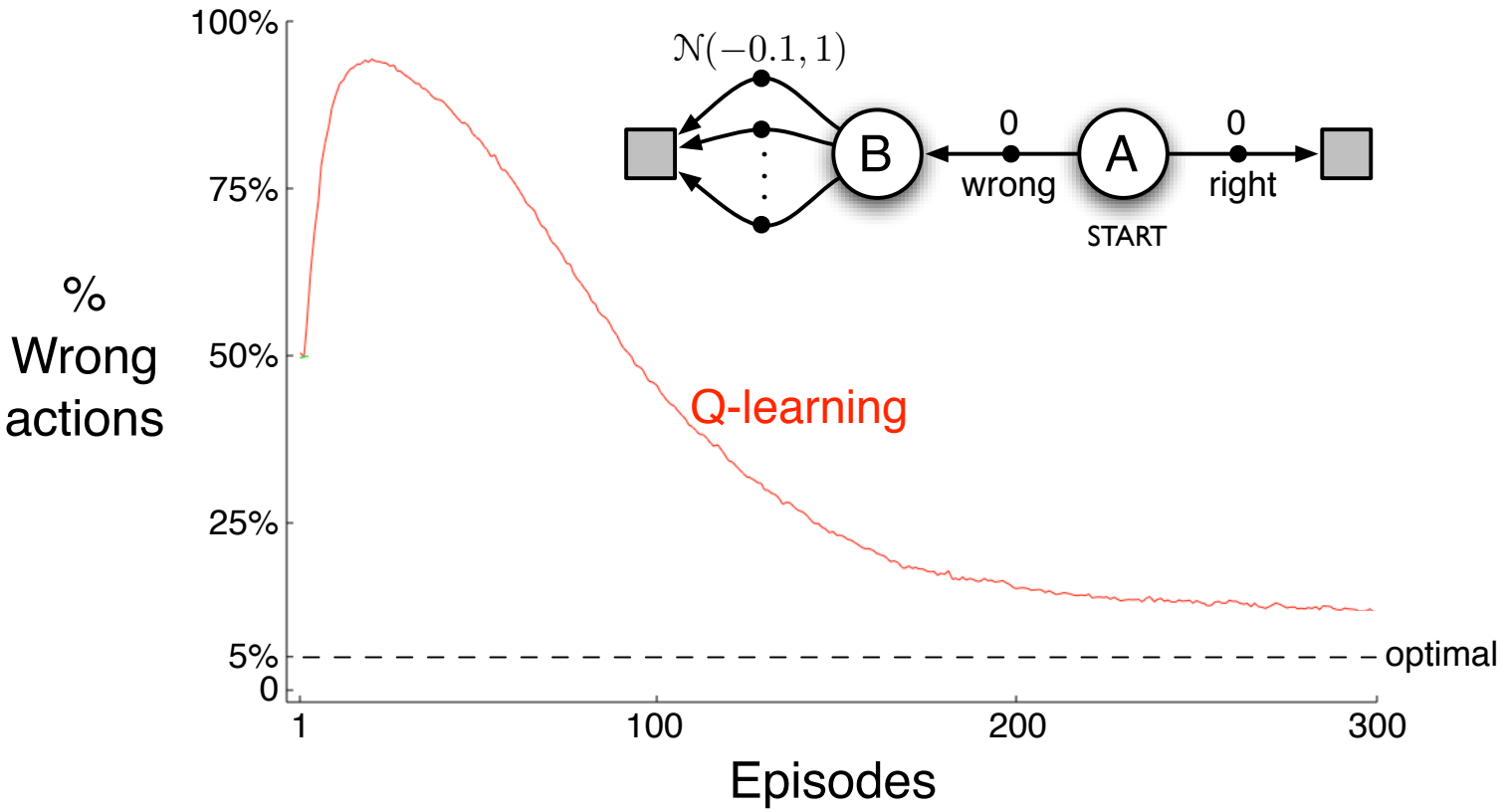
Expected Sarsa

- Expected Sarsa's performs better than Sarsa (but costs more)

Performance on the Cliff-walking Task



Maximization Bias Example



Tabular Q-learning: $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Hado van Hasselt 2010

Double Q-Learning

- Train 2 action-value functions, Q_1 and Q_2
- Do Q-learning on both, but
  - never on the same time steps (Q_1 and Q_2 are indep.)
  - pick Q_1 or Q_2 at random to be updated on each step
- If updating Q_1, use Q_2 for the value of the next state:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big(R_{t+1} + Q_2\big(S_{t+1}, \operatorname*{arg\,max}_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big)$$

- Action selections are (say) ε-greedy with respect to the sum of Q_1 and Q_2

Hado van Hasselt 2010

Double Q-Learning

Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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:

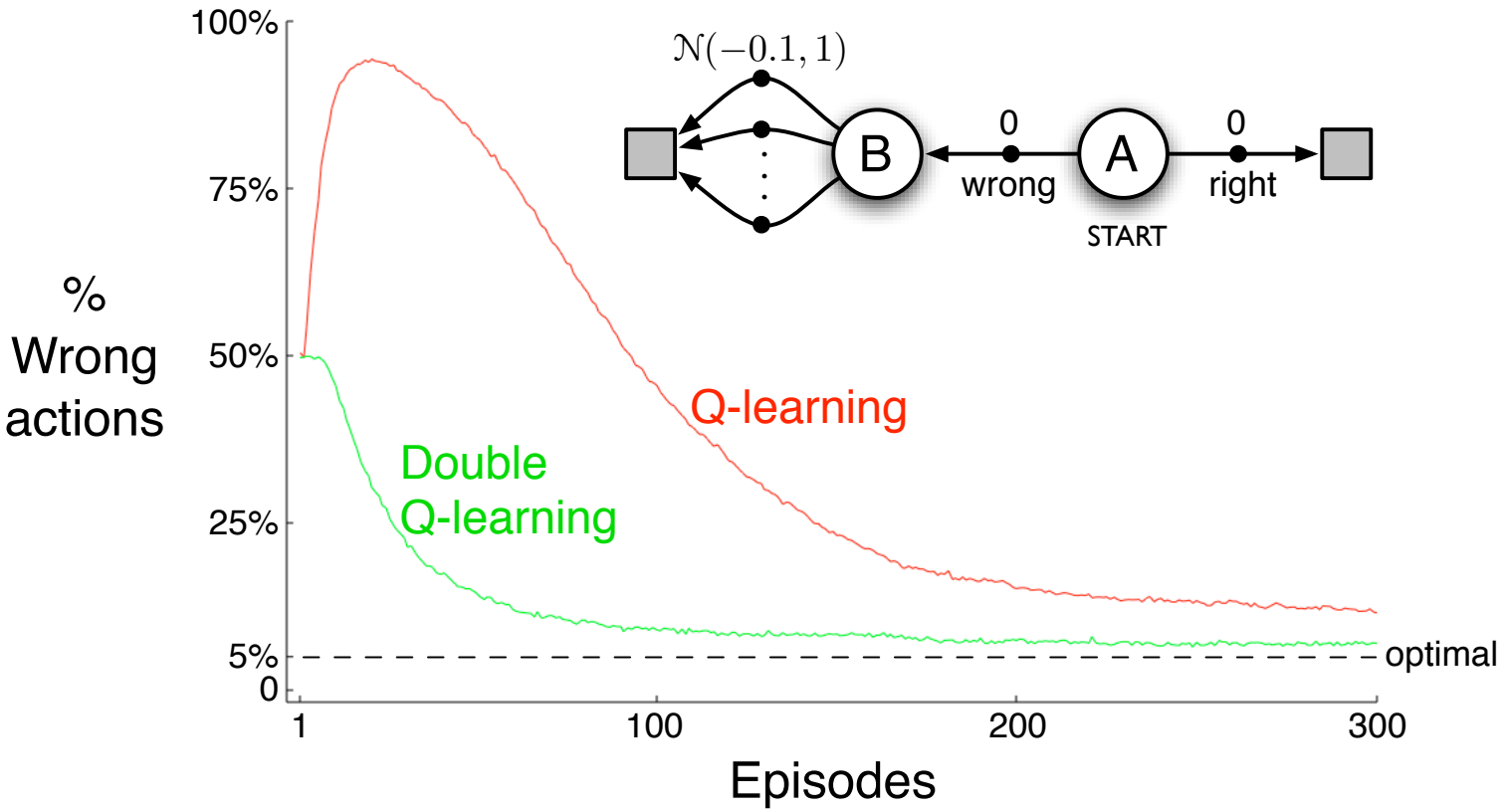
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    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

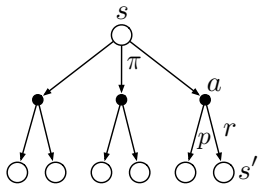
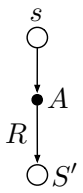
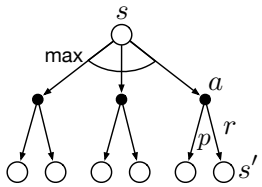
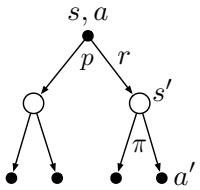
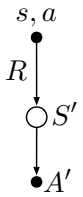
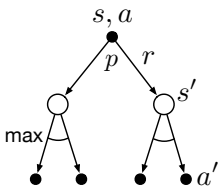
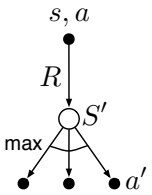
Example of Maximization Bias



Double Q-learning:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big]$$

Expected and Sample Backups (One-Step)

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$v_\pi(s)$	 policy evaluation	 TD(0)
$v_*(s)$	 value iteration	
$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
$q_*(s,a)$	 q-value iteration	 Q-learning

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 congenial*
- 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