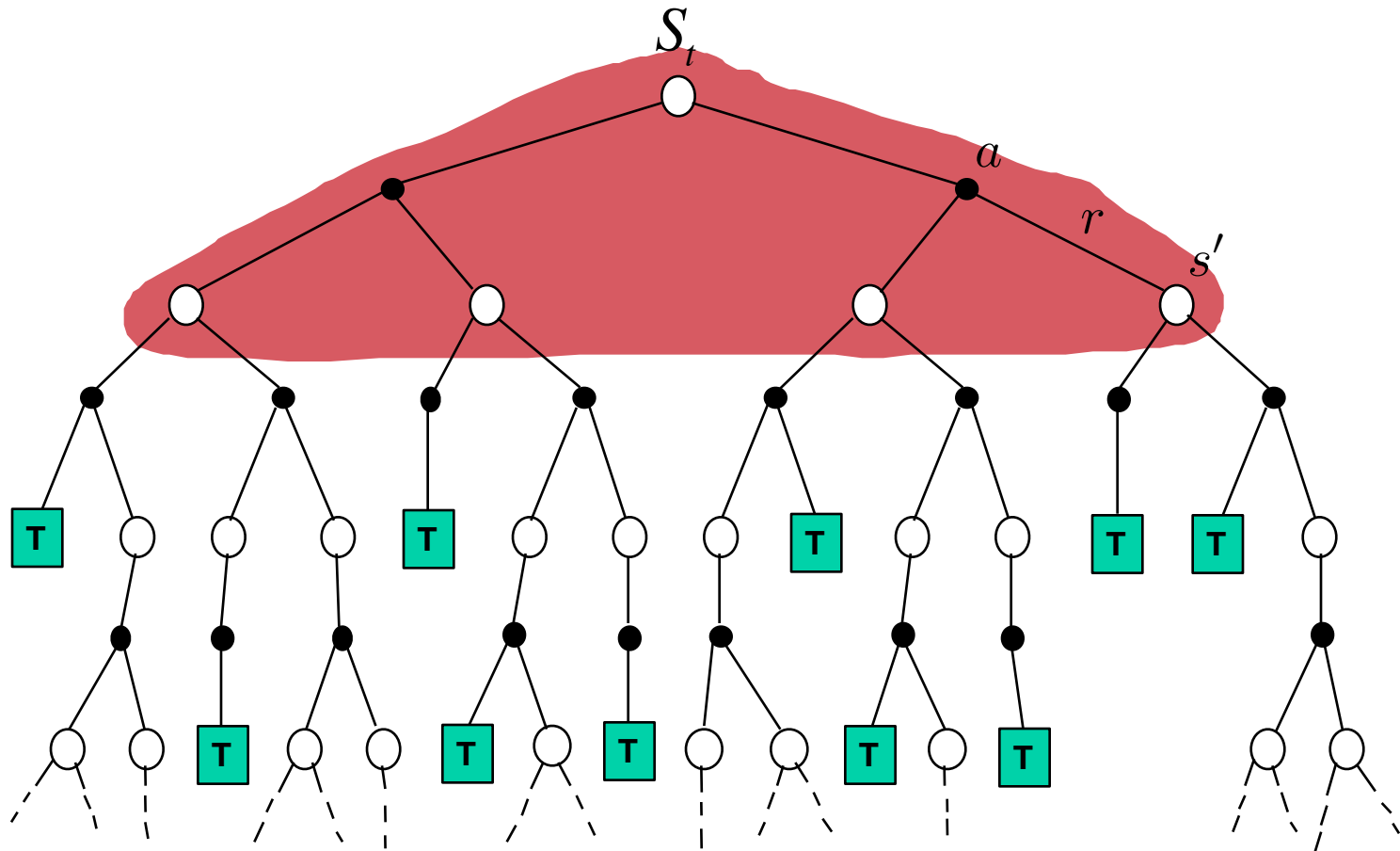


Sequential decision making  
Wrap-up of policy evaluation  
Control: MC control, Sarsa, Q-learning

# Recall: DP Policy Evaluation

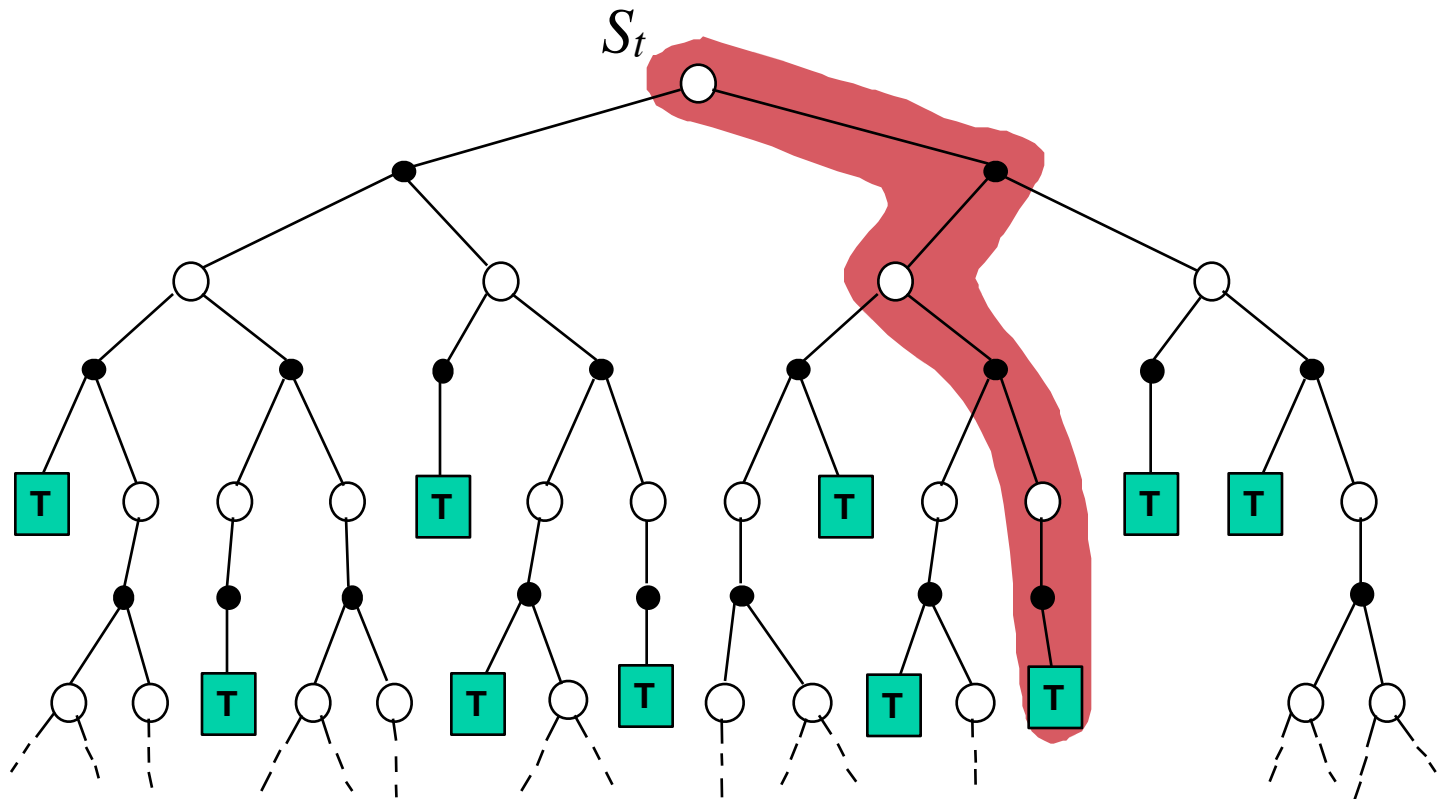
$$V(S_t) \leftarrow E_{\pi} [R_{t+1} + \gamma V(S_{t+1})] = \sum_a \pi(a|S_t) \sum_{s', r} p(s', r|S_t, a) [r + \gamma V(s')]$$



# Recall: Simple Monte Carlo

---

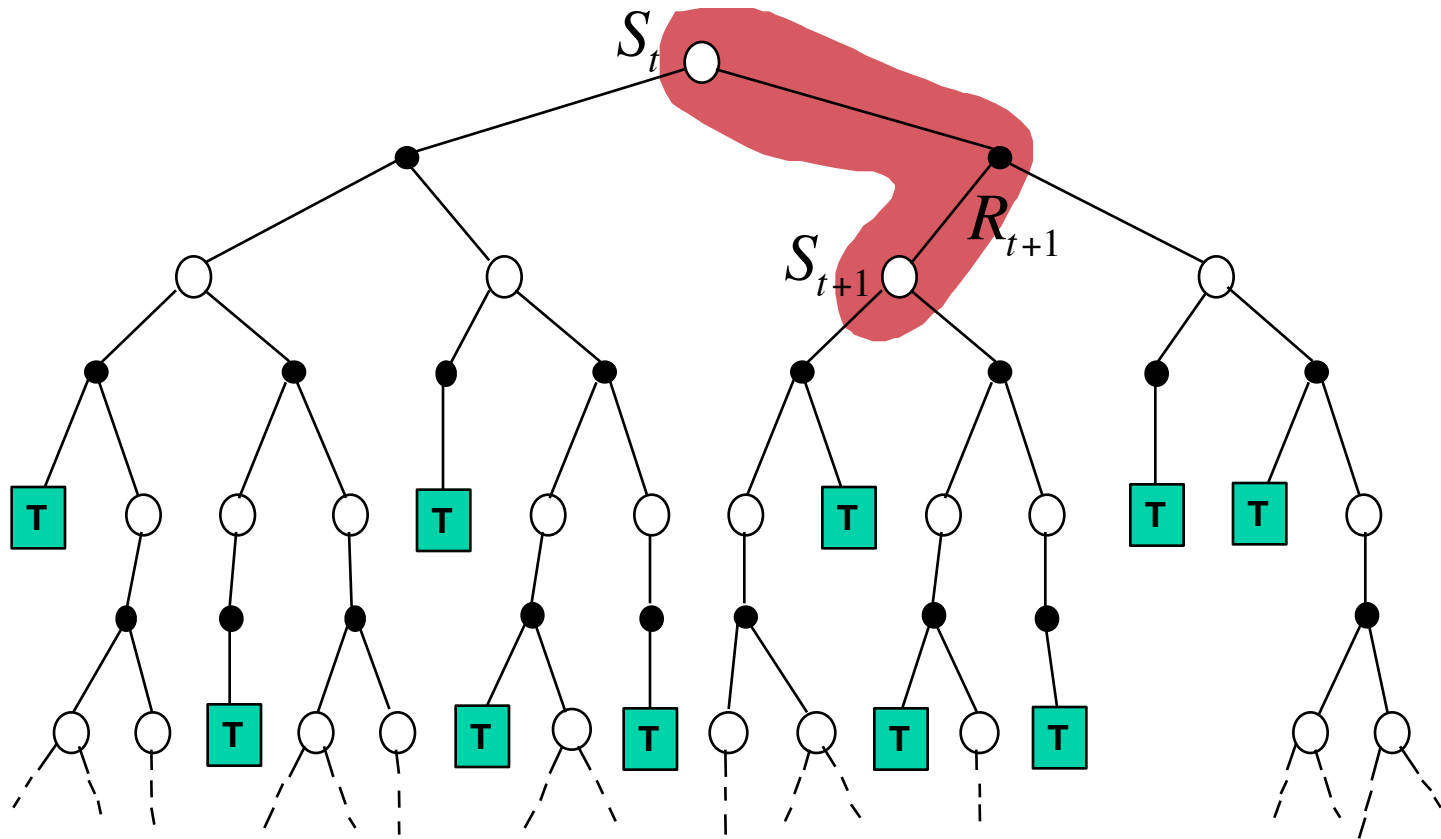
$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$



# Recall: Simplest TD Method (TD(0))

---

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





# Recall: TD Prediction

---

## Policy Evaluation (the prediction problem):

for a given policy  $\pi$ , compute the state-value function  $v_\pi$

Recall: Simple every-visit Monte Carlo method:

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

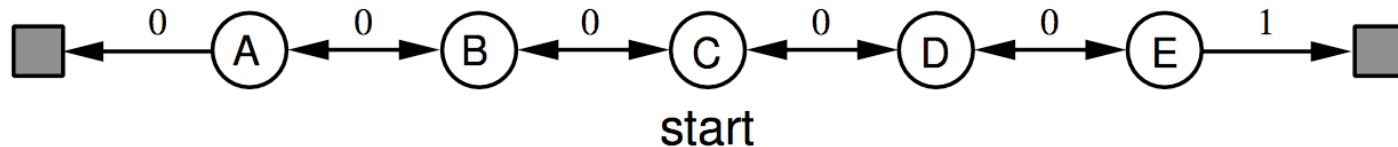
 **target**: the actual return after time  $t$

The simplest temporal-difference method TD(0):

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

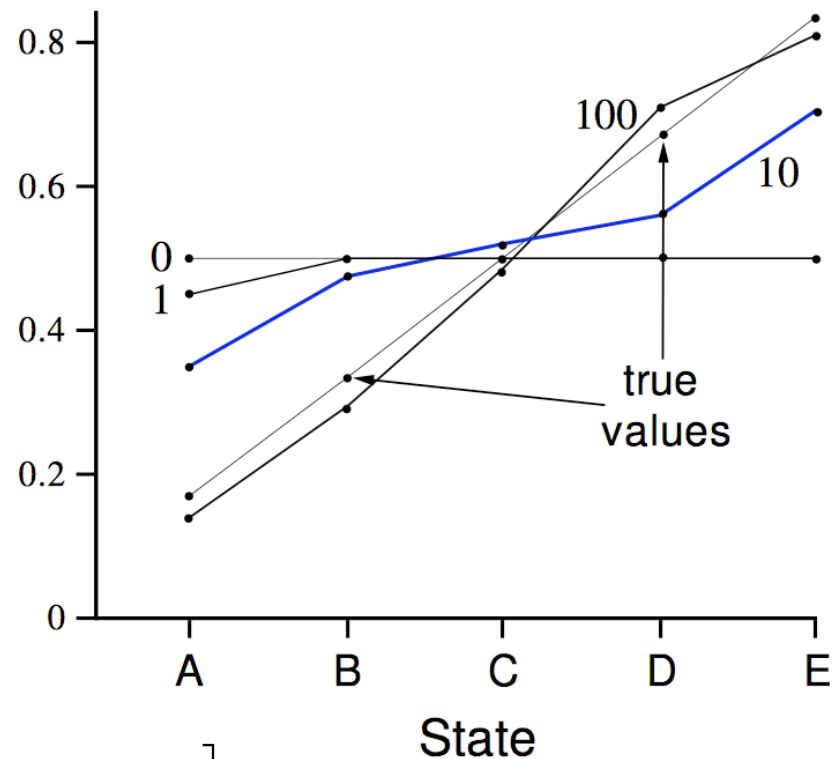
**target**: an estimate of the return

# Random Walk Example



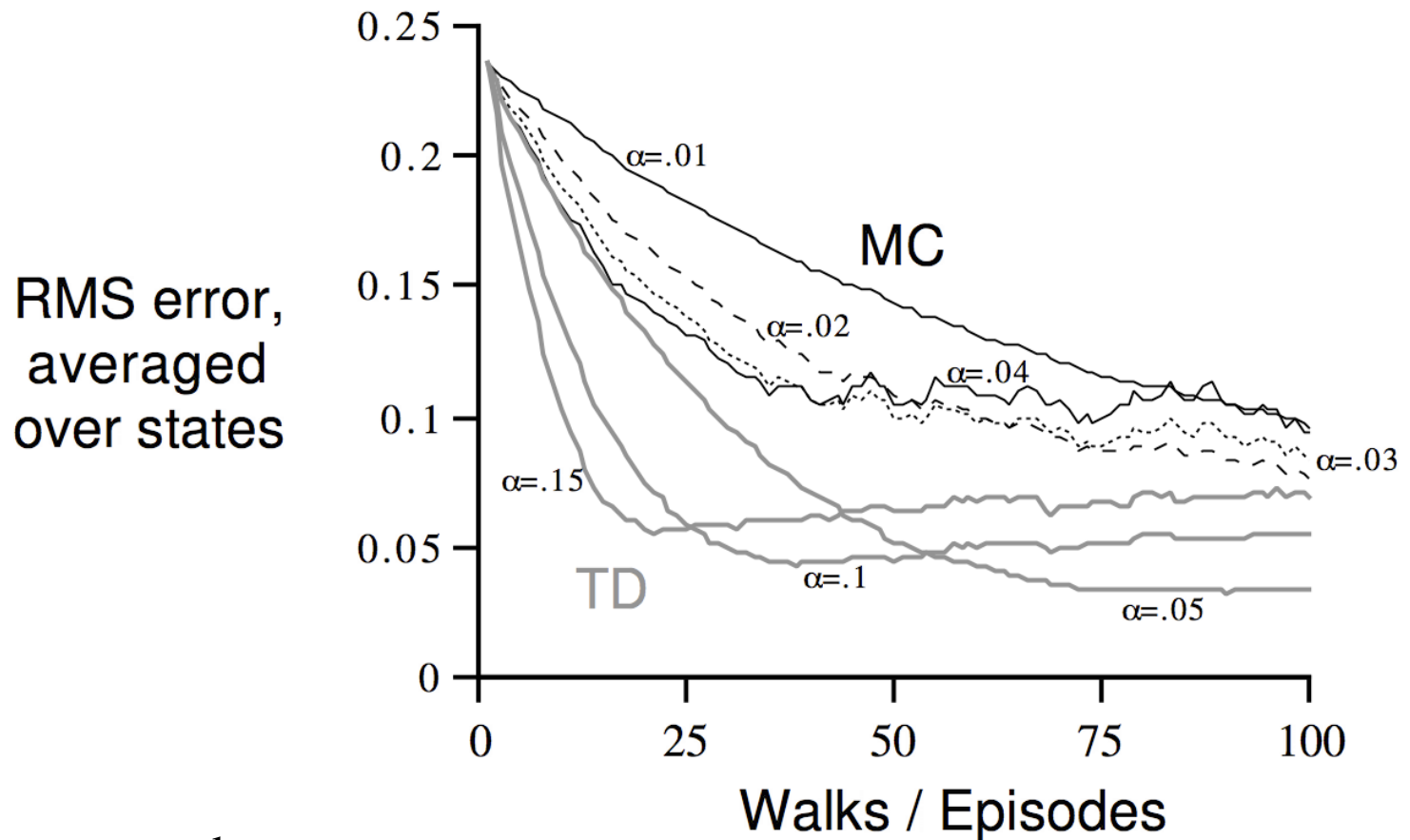
Values learned by TD after various numbers of episodes

Estimated value



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

# 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.

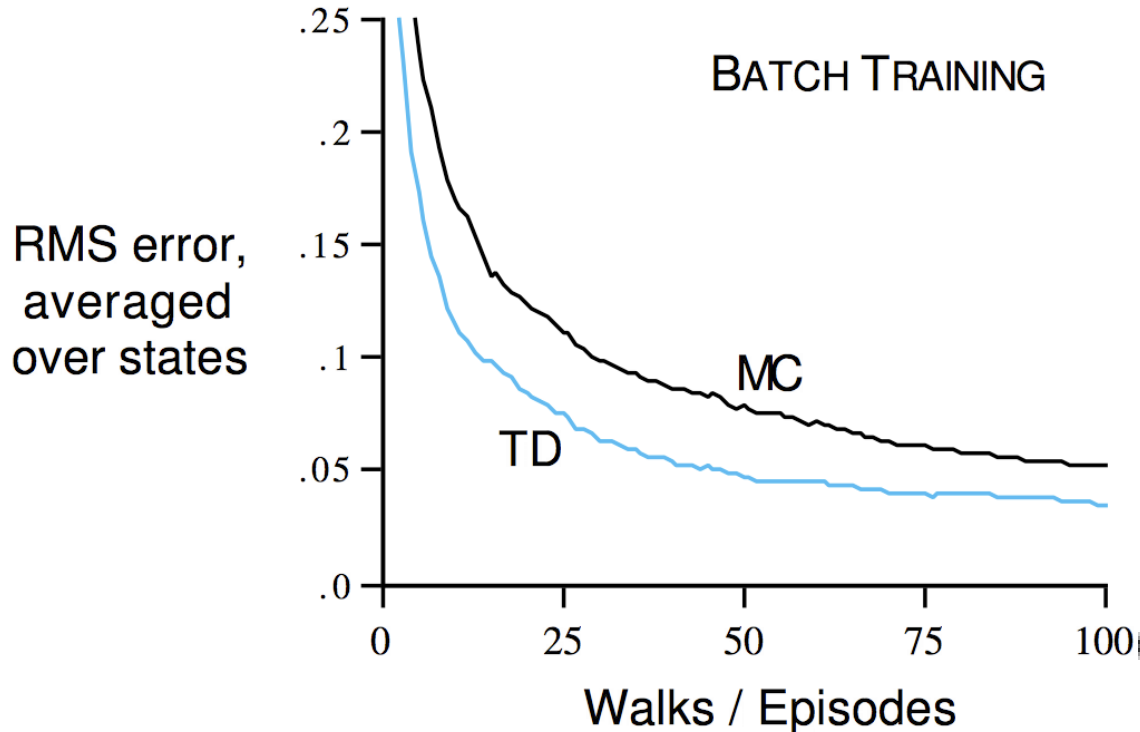
Compute updates according to TD or MC, but only update estimates after each complete pass through the data.

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

Constant- $\alpha$  MC also converges under these conditions, **but to a different answer!**

# Random Walk under Batch Updating

---



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

# You are the Predictor

---

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

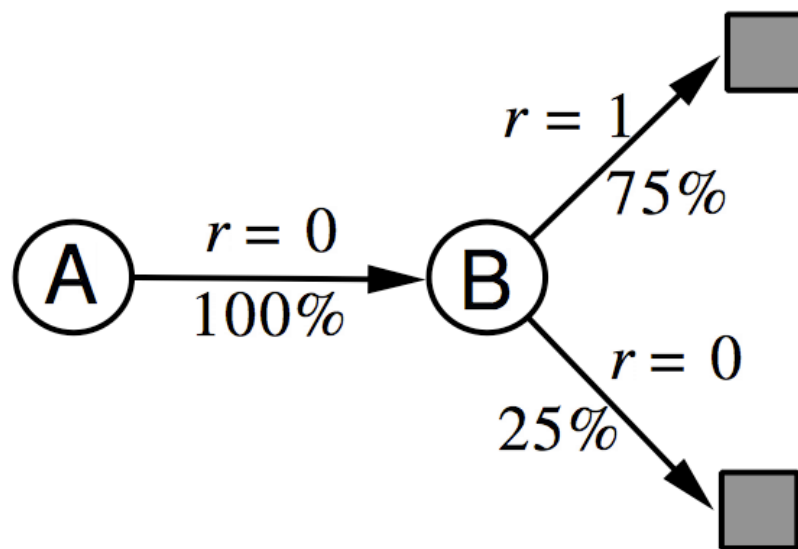
$V(B)?$

$V(A)?$

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

# You are the Predictor

---



$V(A)?$

# You are the Predictor

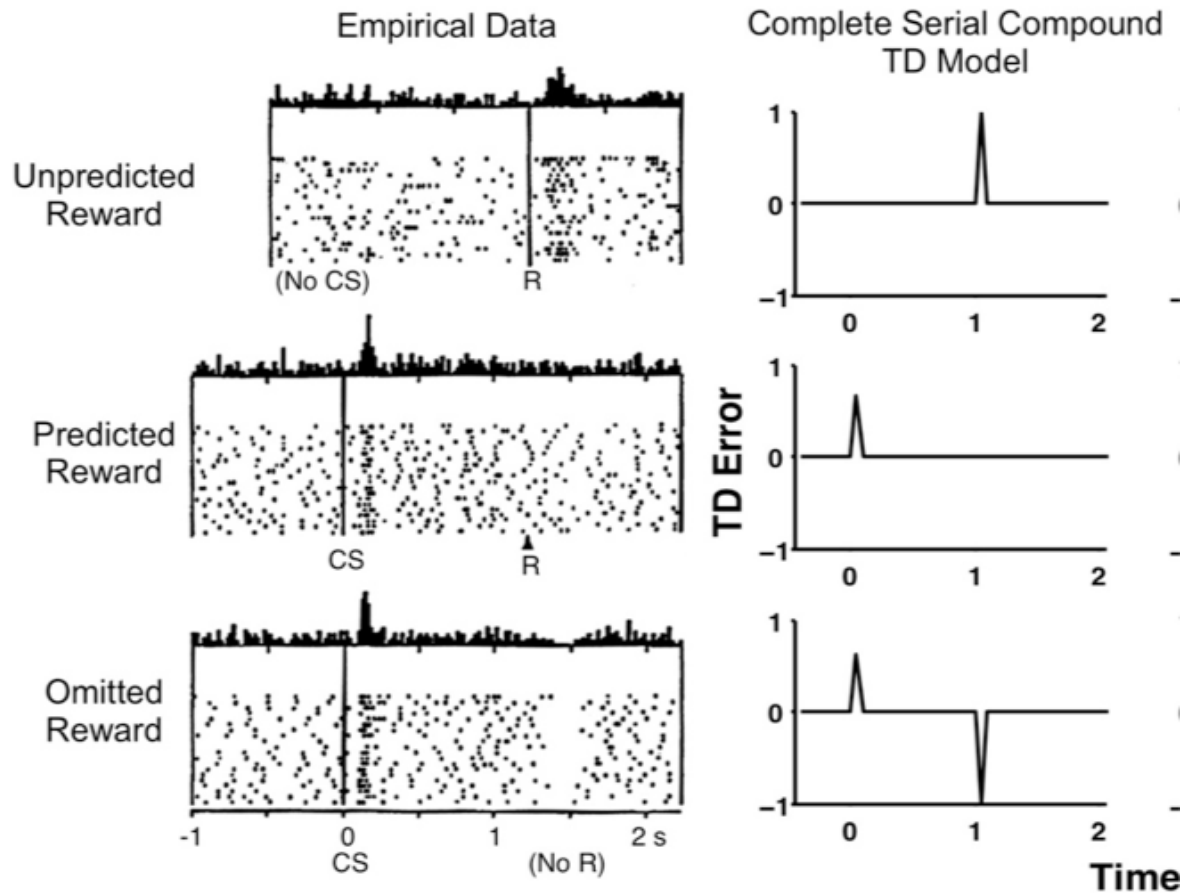
---

- The prediction that best matches the training data is  $V(A)=0$ 
  - This **minimizes the mean-square-error** on the training set
  - This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set  $V(A)=.75$ 
  - This is correct for the **maximum likelihood** estimate of a Markov model generating the data
  - i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts (how?)
  - This is called the **certainty-equivalence estimate**
  - This is what TD gets



# Application of TD

## Dopamine neuron activity modelling



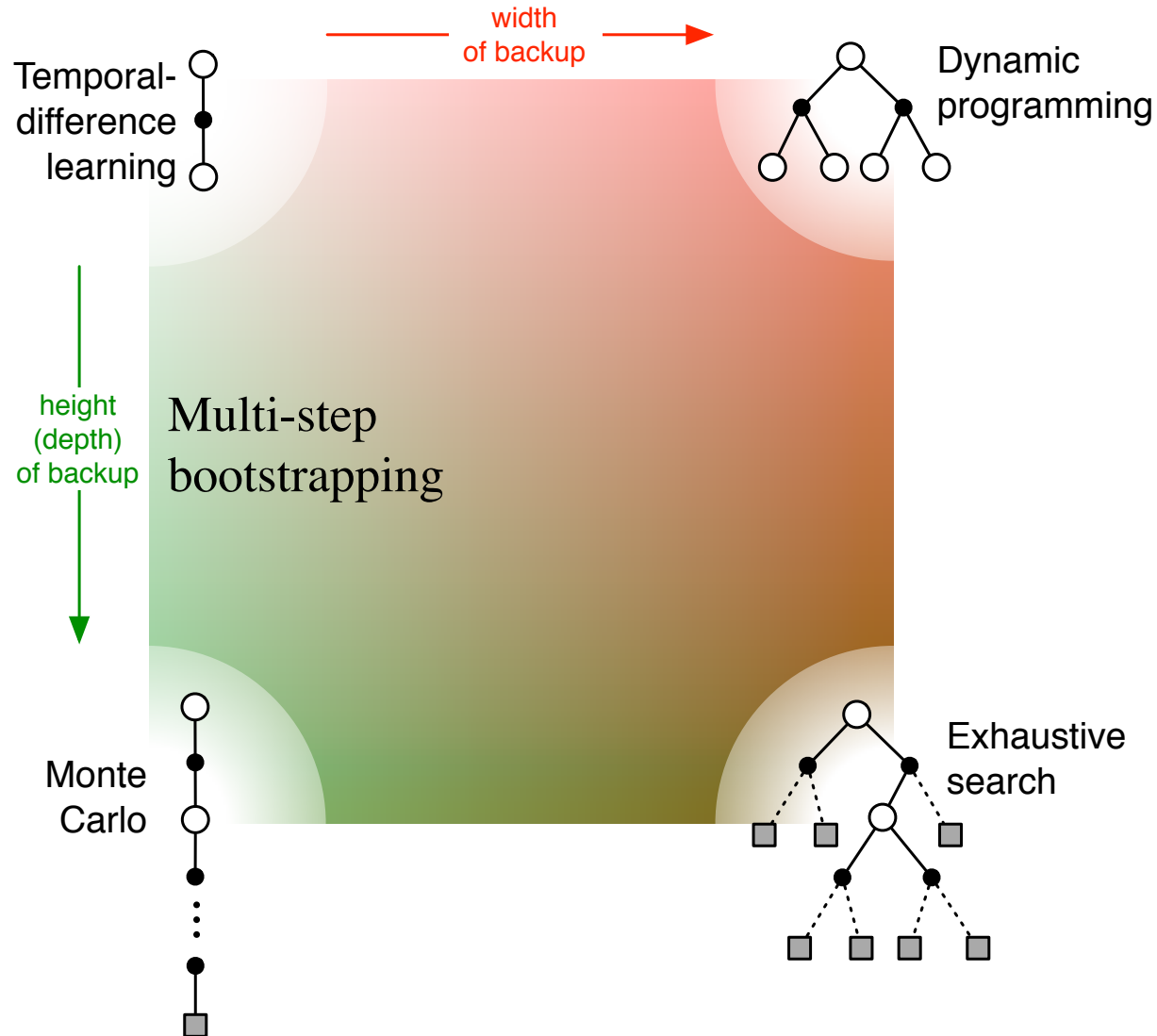
Cf. Schultz, Dayan et al, 1996; and lots of follow-up work including MNI, Psych.

# 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



# $n$ -step TD Prediction

1-step TD  
and TD(0)



2-step TD



3-step TD



...

$n$ -step TD



...

$\infty$ -step TD  
and Monte Carlo



Idea: Look farther into the future when you do TD — backup (1, 2, 3, ...,  $n$  steps)

# Mathematics of $n$ -step TD Returns/Targets

---

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

- **TD:**  $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

- Use  $V_t$  to estimate remaining return

- **$n$ -step TD:**

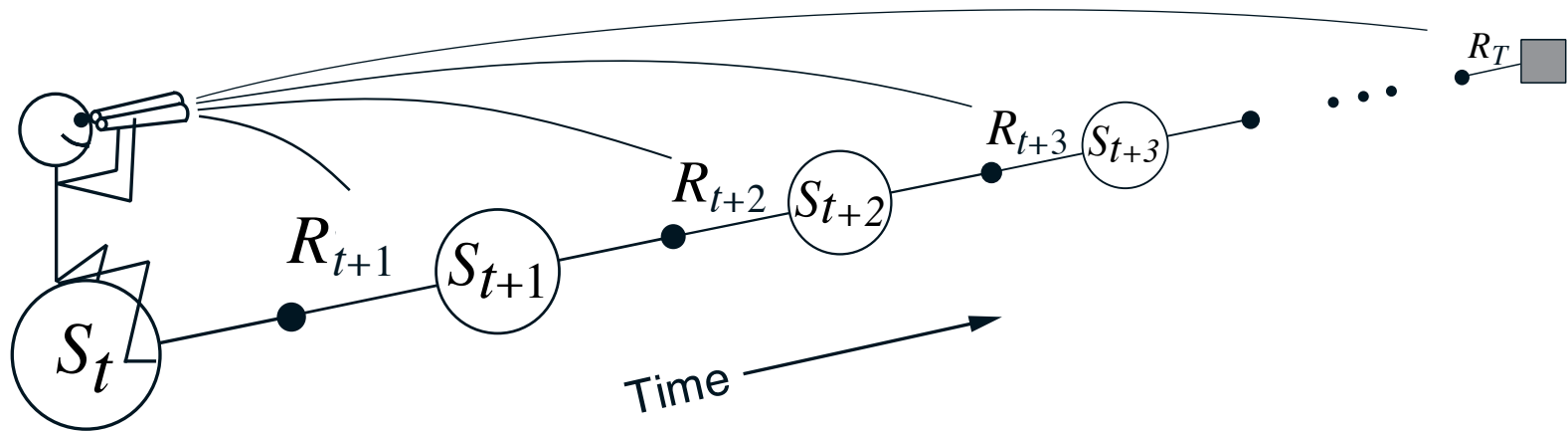
- 2 step return:  $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$

- $n$ -step return:  $G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_t(S_{t+n})$

with  $G_t^{(n)} \doteq G_t$  if  $t + n \geq T$

# Forward View

- Look forward from each state to determine update from future states and rewards:



# *n*-step TD

---

- Recall the *n*-step return:

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

- Of course, this is not available until time *t+n*
- The natural algorithm is thus to **wait** until then:

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

- This is called *n*-step TD

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

Initialize  $V(s)$  arbitrarily,  $s \in \mathcal{S}$

Parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$

All store and access operations (for  $S_t$  and  $R_t$ ) can take their index mod  $n$

Repeat (for each episode):

    Initialize and store  $S_0 \neq$  terminal

$T \leftarrow \infty$

    For  $t = 0, 1, 2, \dots$  :

        | 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  $T \leftarrow t + 1$

        |  $\tau \leftarrow t - n + 1$     ( $\tau$  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:  $G \leftarrow G + \gamma^n V(S_{\tau+n})$   $(G_\tau^{(n)})$

            |  $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$

    Until  $\tau = T - 1$



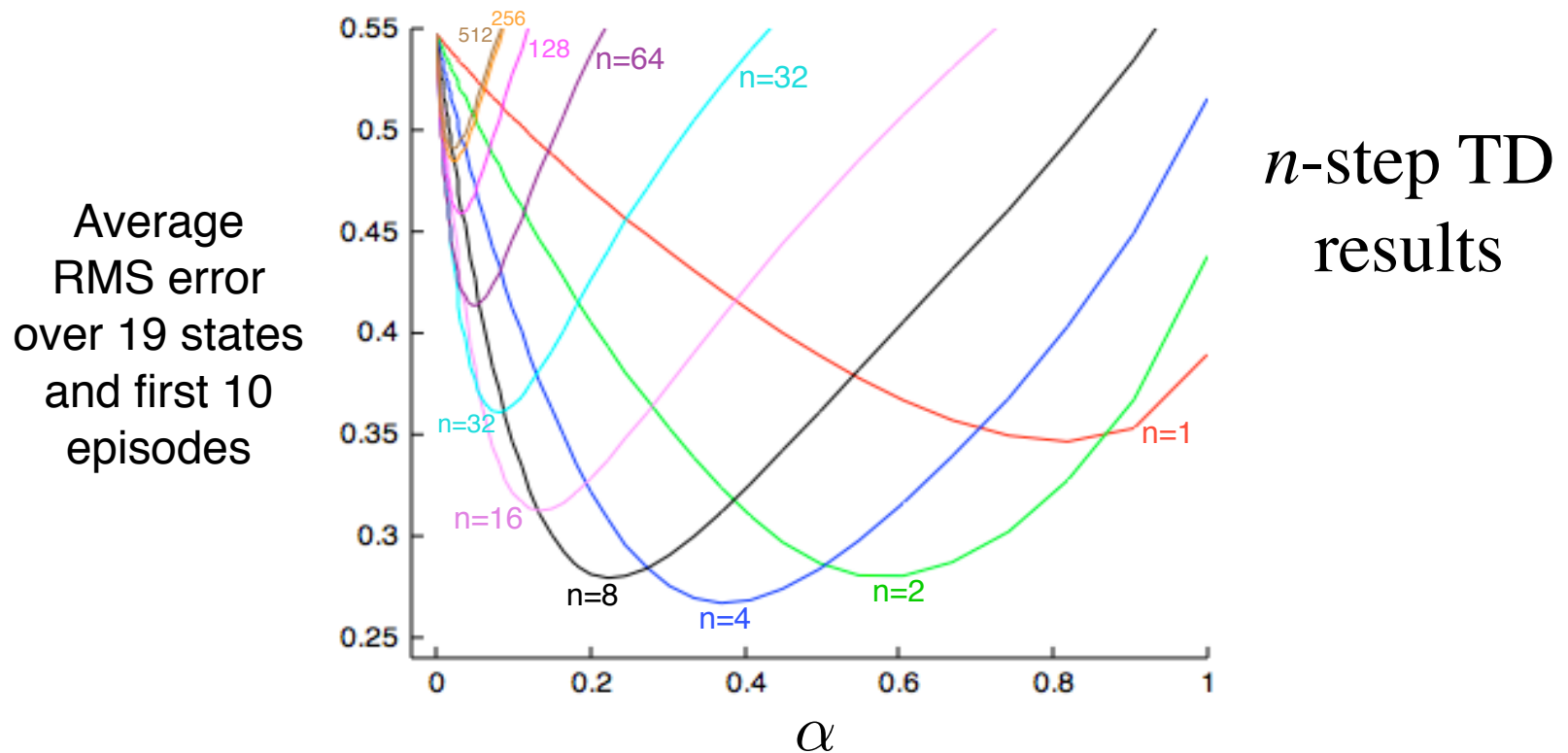
# Random Walk Examples

---



- How does 2-step TD work here?
- How about 3-step TD?

# A Larger Example – 19-state Random Walk



- An intermediate  $\alpha$  is best
- An intermediate  $n$  is best
- Do you think there is an optimal  $n$ ? for every task?

# Conclusions Regarding $n$ -step Methods (so far)

---

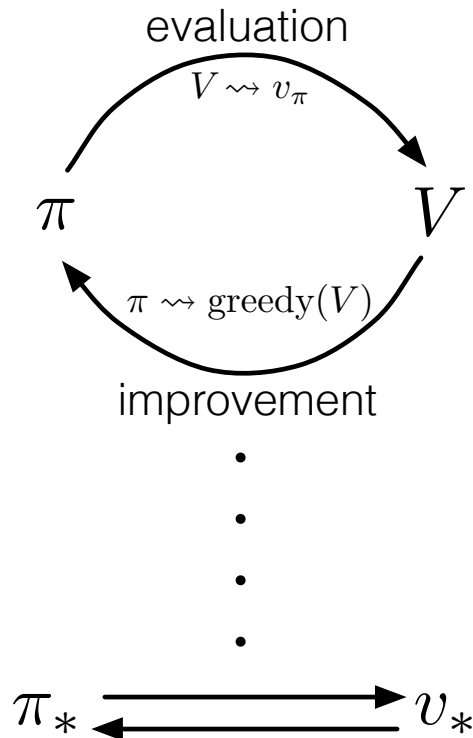
- Generalize Temporal-Difference and Monte Carlo learning methods, sliding from one to the other as  $n$  increases
  - $n = 1$  is TD(0)  $n = \infty$  is MC
  - an intermediate  $n$  is often much better than either extreme
  - applicable to both continuing and episodic problems
- There is some cost in computation
  - need to remember the last  $n$  states
  - learning is delayed by  $n$  steps
  - per-step computation is small and uniform, like TD

# How to do control? GPI!

---

## Generalized Policy Iteration (GPI):

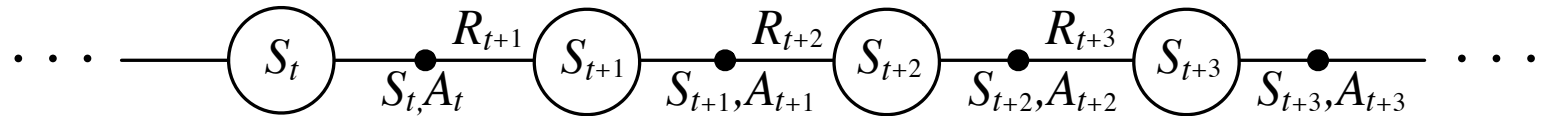
any interaction of policy evaluation and policy improvement, independent of their granularity.



# Monte Carlo Estimation of Action Values

---

Estimate  $q_\pi$  for the current policy  $\pi$



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$$

$$\text{where } G_t = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k}$$

and  $T$  is the time of entering terminal state

# Monte Carlo Estimation of Action Values (Q)

---

- ❑  $q_{\pi}(s,a)$  - average return starting from state  $s$  and action  $a$  following  $\pi$
- ❑ Converges asymptotically *if* every state-action pair is visited
- ❑ *Exploring starts*: Every state-action pair has a non-zero probability of being the starting pair

# On-policy Monte Carlo Control

---

- ❑ *On-policy*: learn about policy currently executing
- ❑ How do we get rid of exploring starts?
  - The policy must be eternally *soft*:
    - $\pi(a|s) > 0$  for all  $s$  and  $a$
  - e.g.  $\epsilon$ -soft policy:
    - probability of an action =  $\frac{\epsilon}{|\mathcal{A}(s)|}$  or  $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$   
non-max                      max (greedy)
- ❑ Similar to GPI: move policy *towards* greedy policy (e.g.,  $\epsilon$ -greedy)
- ❑ Converges to best  $\epsilon$ -soft policy

# On-policy MC Control

---

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

$Q(s, a) \leftarrow$  arbitrary

$Returns(s, a) \leftarrow$  empty list

$\pi(a|s) \leftarrow$  an arbitrary  $\varepsilon$ -soft policy

Repeat forever:

(a) Generate an episode using  $\pi$

(b) For each pair  $s, a$  appearing in the episode:

$G \leftarrow$  return following the first occurrence of  $s, a$

Append  $G$  to  $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each  $s$  in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

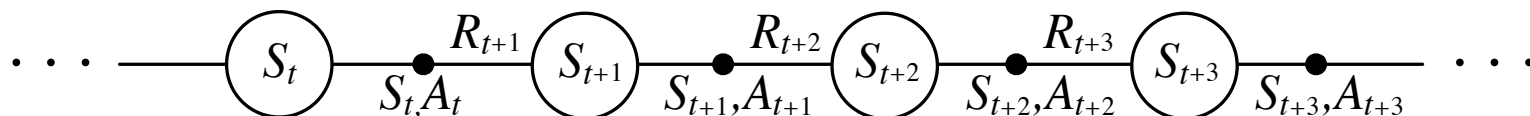
$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$



# TD-Style Learning for Action-Values

---

Estimate  $q_\pi$  for the current policy  $\pi$



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

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

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

# Sarsa: On-Policy TD Control

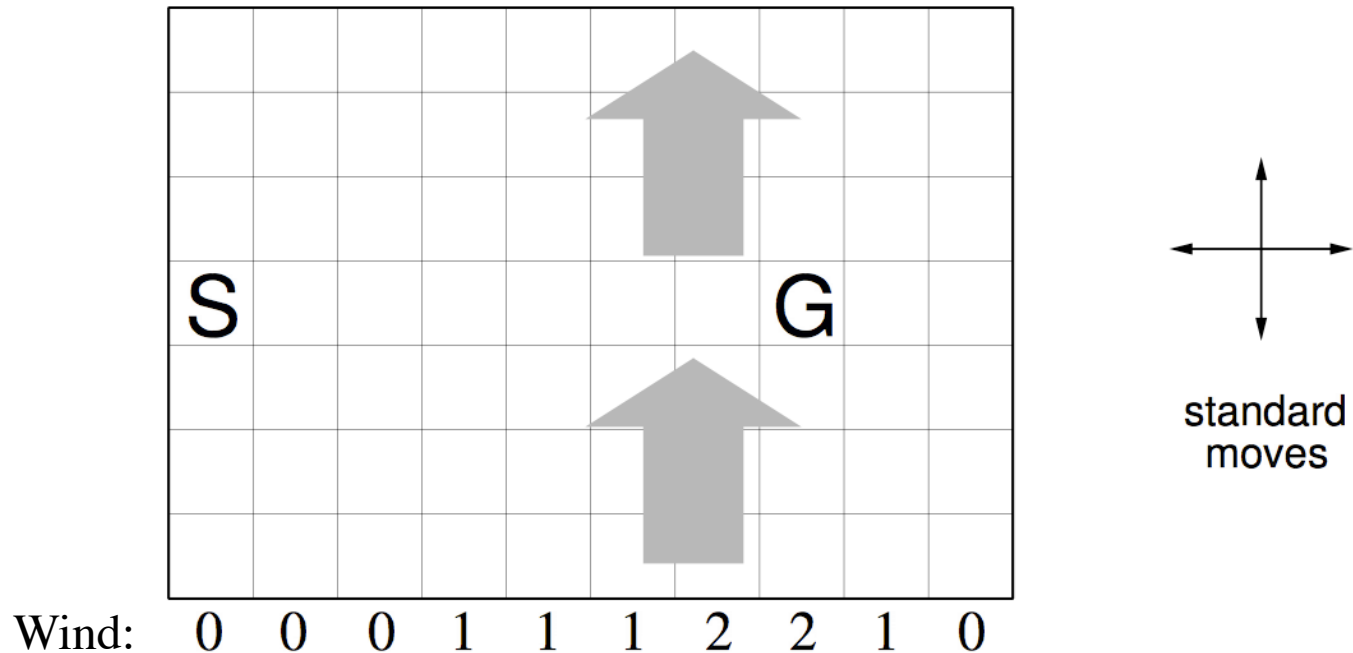
---

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  $Q(\text{terminal-state}, \cdot) = 0$ 
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)
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A';$ 
  until  $S$  is terminal
```

# Windy Gridworld

---



undiscounted, episodic, reward =  $-1$  until goal

The main graph illustrates the total number of episodes completed as a function of time steps. The x-axis represents 'Time steps' from 0 to 8000, and the y-axis represents 'Episodes' from 0 to 170. The curve shows a steady increase in episodes over time, reaching approximately 170 episodes by 8000 time steps.

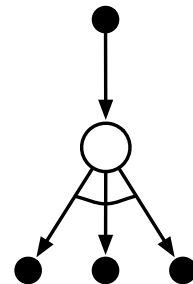
The inset graph shows a sequence of states (S and G) and actions (0 and 1) over time steps 0 to 6. The inset shows a sequence of states (S and G) and actions (0 and 1) over time steps 0 to 6. The inset shows a sequence of states (S and G) and actions (0 and 1) over time steps 0 to 6.

# Q-Learning: Off-Policy TD Control

---

One-step Q-learning:

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



Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize  $S$

Repeat (for each step of episode):

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

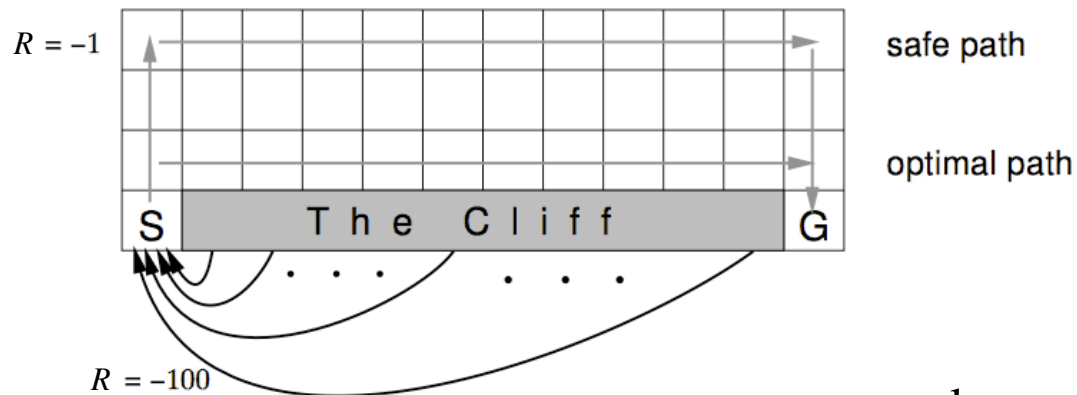
Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

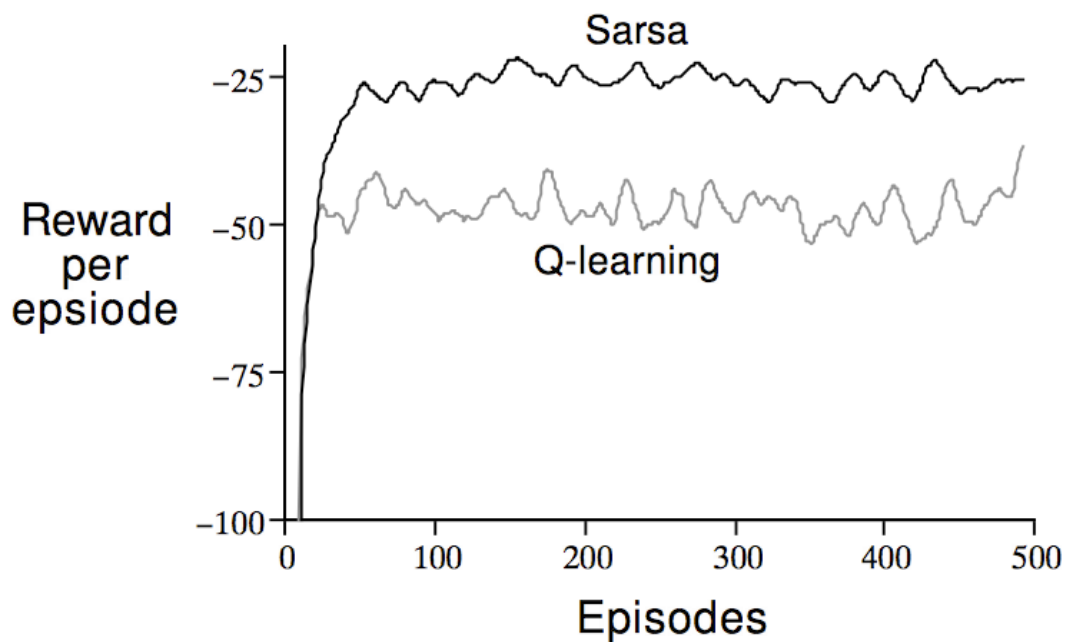
$S \leftarrow S'$ ;

until  $S$  is terminal

# Cliffwalking



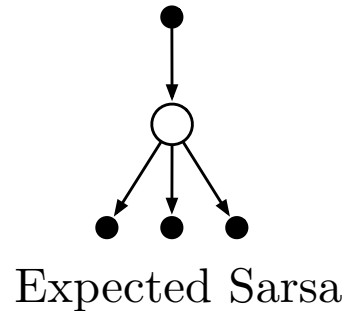
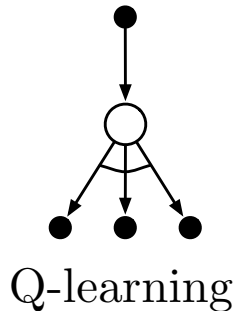
$\epsilon$ -greedy,  $\epsilon = 0.1$



# Expected Sarsa

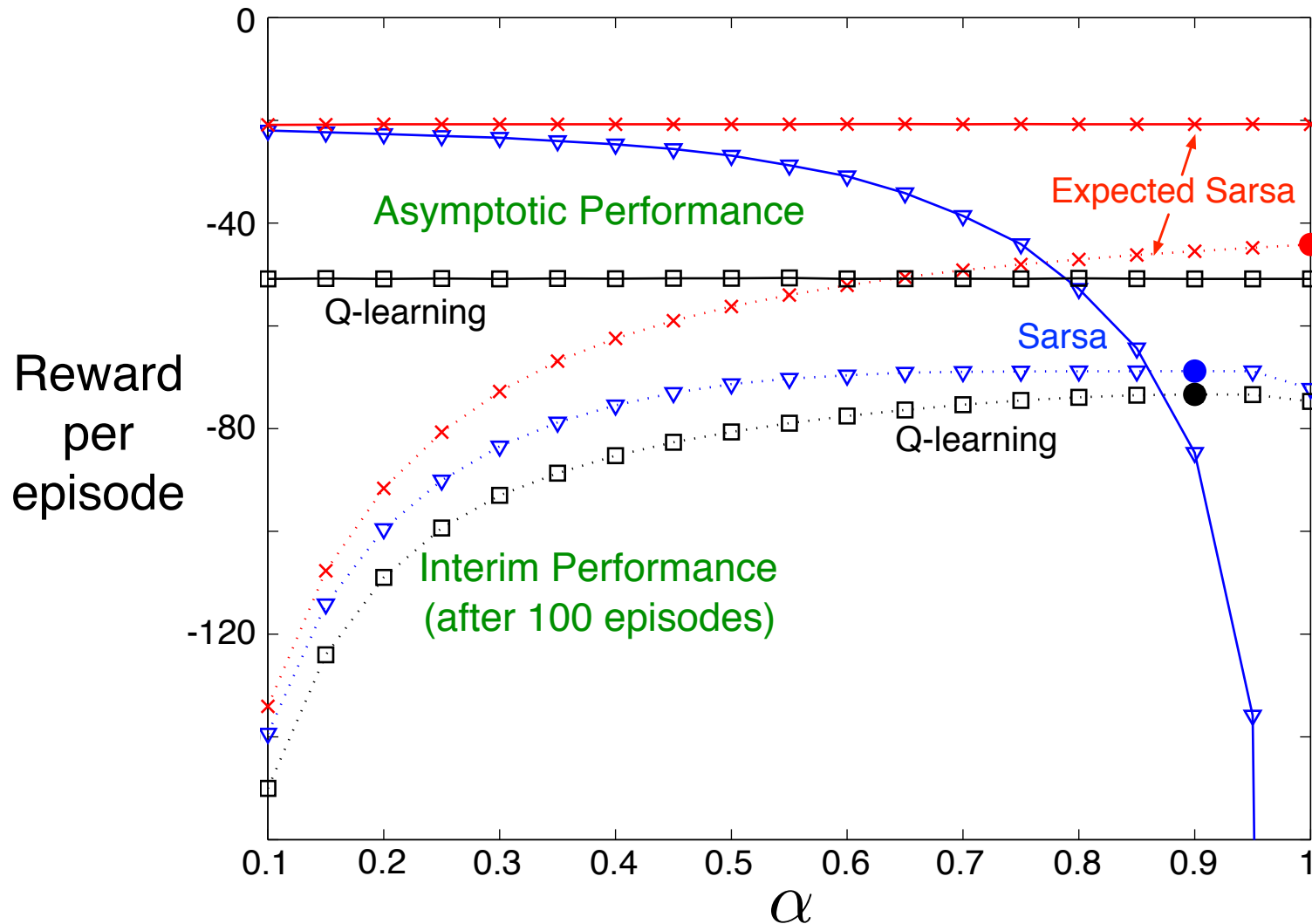
- Instead of the *sample* value-of-next-state, use the expectation!

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



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

# Performance on the Cliff-walking Task



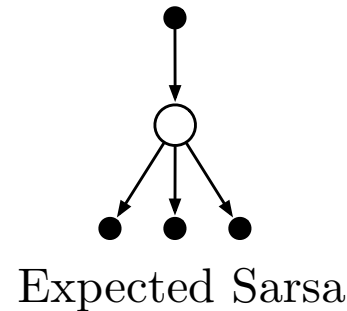
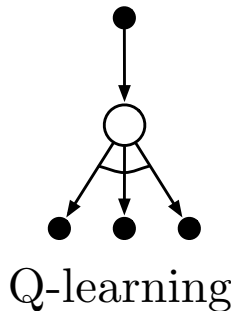


# Off-policy Expected Sarsa

- Expected Sarsa generalizes to arbitrary behavior policies  $\mu$ 
  - in which case it includes Q-learning as the special case in which  $\pi$  is the greedy policy

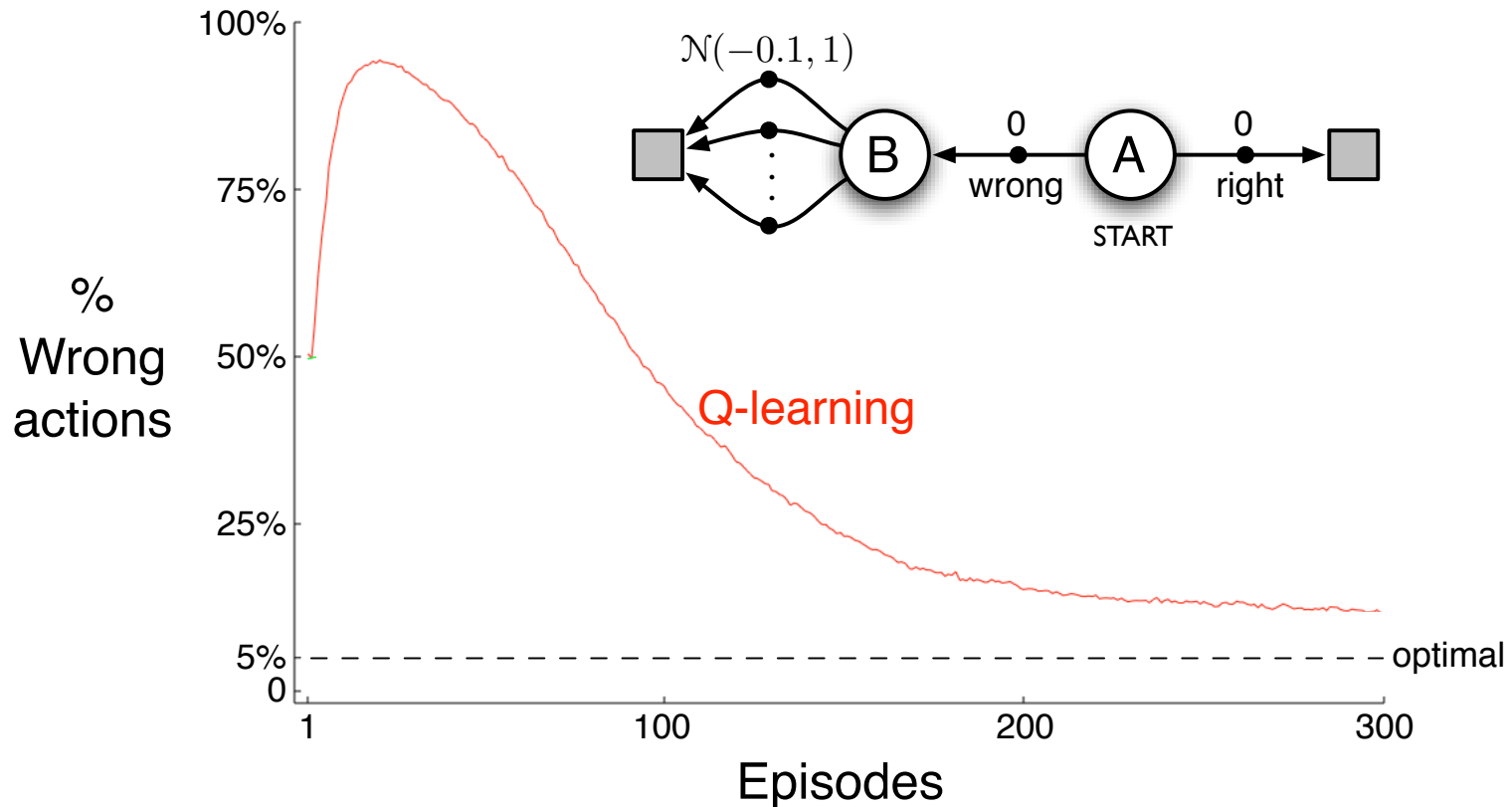
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right]$$
$$\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Nothing  
changes  
here



- This idea seems to be new

# Maximization Bias Example



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

# 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:
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left( R_{t+1} + Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right)$$
- Action selections are (say)  $\varepsilon$ -greedy with respect to the sum of  $Q_1$  and  $Q_2$

# 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  $Q_1(\text{terminal-state}, \cdot) = Q_2(\text{terminal-state}, \cdot) = 0$

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.,  $\epsilon$ -greedy in  $Q_1 + Q_2$ )

Take action  $A$ , observe  $R, S'$

With 0.5 probability:

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

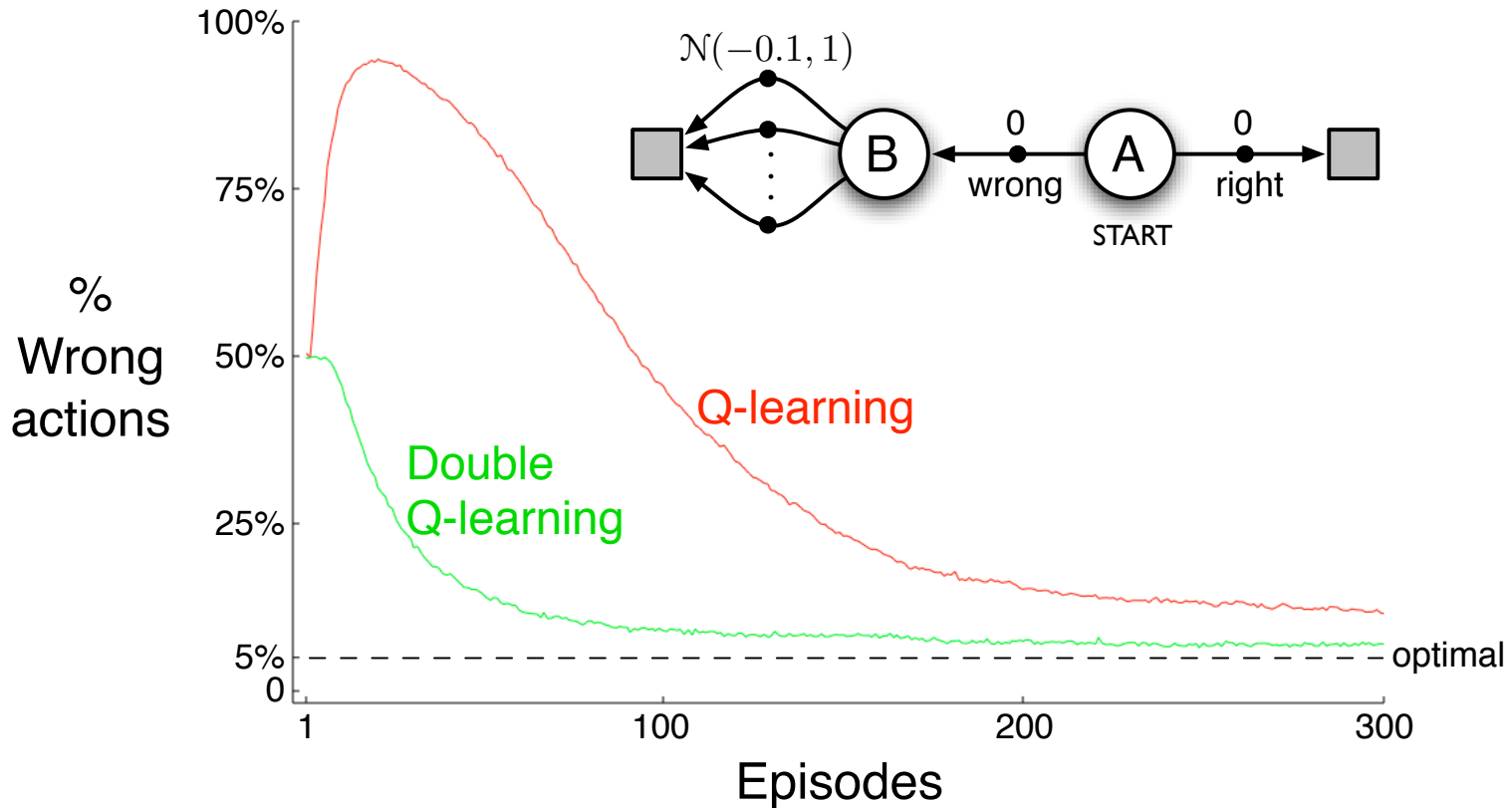
else:

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

$S \leftarrow S'$ ;

until  $S$  is terminal

# Example of Maximization Bias



Double Q-learning:

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

# 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*
- Avoiding maximization bias with Double Q-learning