
# Temporal Difference (TD)

Mohammad **Dehghani**

Mechanical and Industrial Engineering Department

Northeastern University

Feb-20

# Introduction

**Temporal-Difference (TD)** method is a blend of ***Monte Carlo (MC)*** method and ***Dynamic Programming (DP)*** method

### Monte Carlo Method

- **No model:** agent does not know state MDP transitions
- **Sample-based experience:** learn directly from raw experience

### Dynamic Programming

- **Bootstrapping:** update estimates based in part on other learned estimates, without waiting for a final outcome

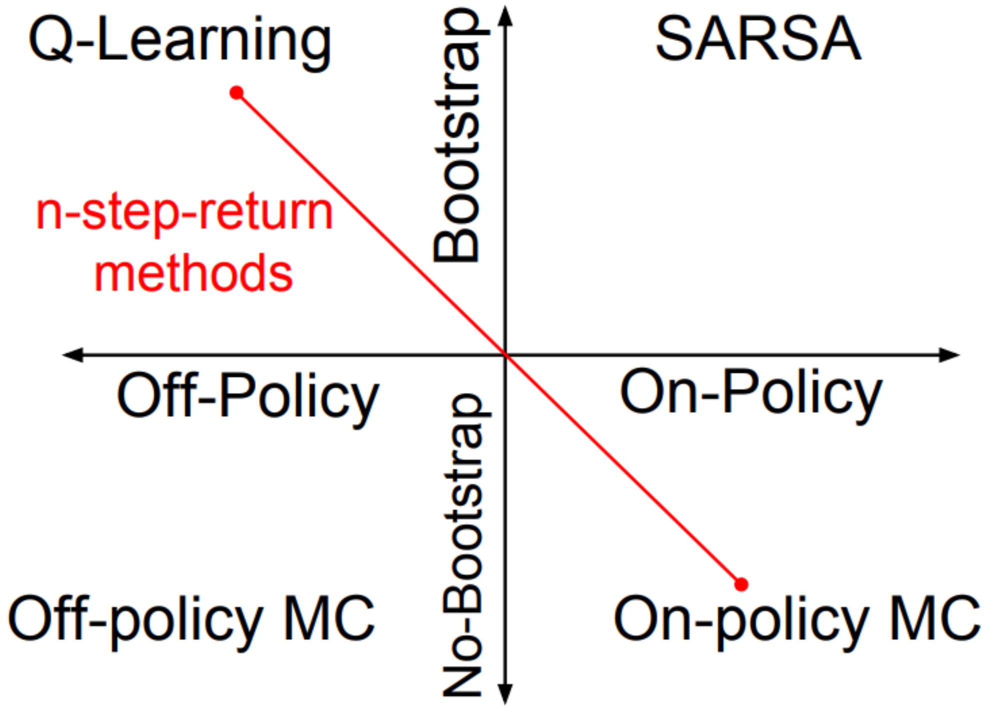
It can learn from **incomplete episode** thus this method can be used in continuous problems as well

TD updates a guess towards a guess and revise the guess based on real experience

## Example: A real life Analogy

- **Student performance evaluation: *Annually* vs. *weekly*??**

**Temporal Difference (TD) learning is the *central* and *novel* theme of reinforcement learning**



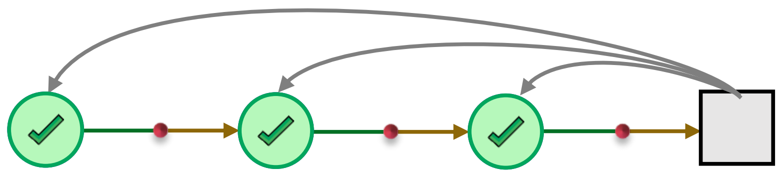
# TD Prediction

## TD vs MCM

- **Both TD and Monte Carlo methods use *experience* to solve the prediction problem.**

**Monte Carlo Methods**

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

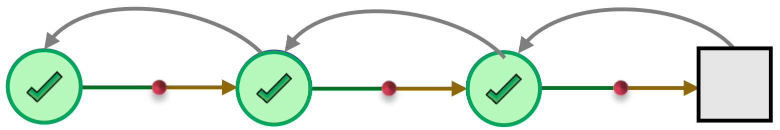


### Episode-by-episode approach:

- Only on the completion of an episode are value estimates and policies changed.
- The agent does mot learn until the end of episode

**Temporal Difference**

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



### Step-by-step approach:

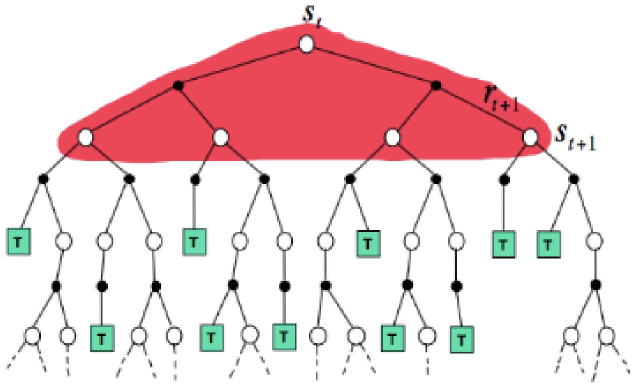
- TD methods need to wait only until the next time step
- State values are updated immediately on transition to a new state
- This method is called ***TD(0)***, or ***one-step TD***

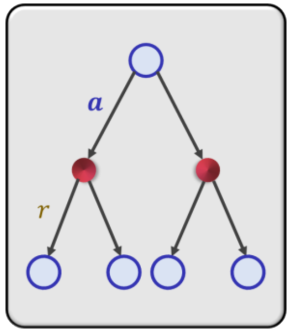
# TD Prediction

## Back up Diagram

### Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$

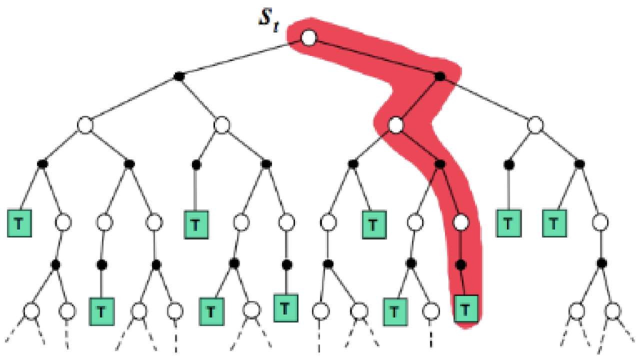


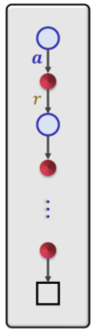




Bootstrapping

### Monte Carlo Methods

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

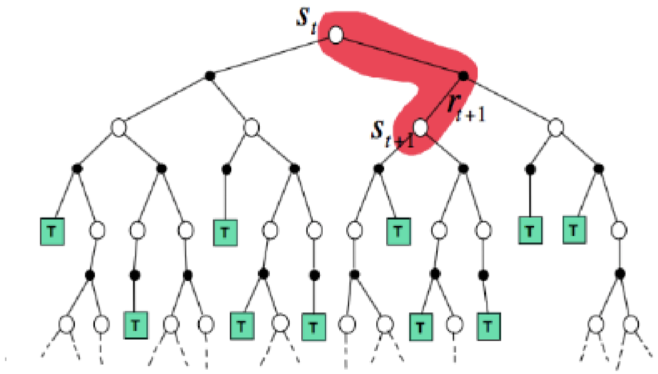


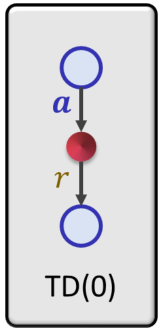



Sampling

### Temporal Difference

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




Bootstrapping & Sampling

# TD Prediction

## Prediction based on the next state value

**Monte Carlo Methods**

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

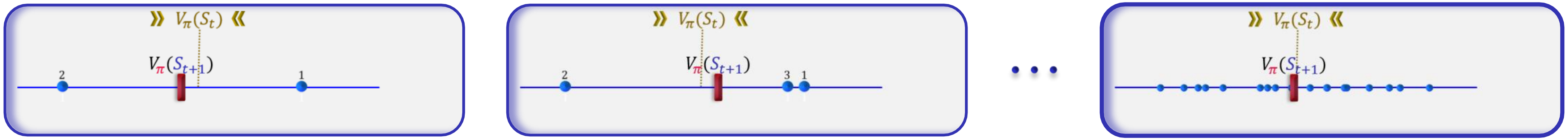
**Target:** the actual return

**Temporal Difference**

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

**Target:** an estimate of the return

- **TD updates the value of one state towards its own estimate of the value in the next state.**
- **Value of the *next state* as a *stand-in* for the return until the end of the episode.**




- **As the estimated value for the next state improves, so does our target.**

- **Bootstrapping:** When you estimate something based on another estimation.

**Dynamic Programming (DP)**

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

All possible next state

**Temporal-Difference (TD)**

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

Just the next state

$$S, A, R, S, A, R, S, A \cdots$$

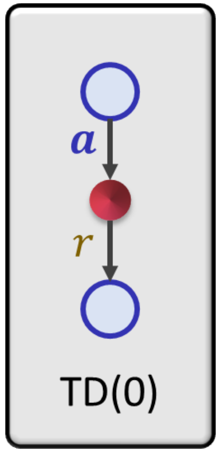
# TD Prediction

## Pseudocode: Tabular TD(0)

**Tabular TD(0) for estimating $v_\pi$**

Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\boxed{\alpha \in (0,1]}$ Learning rate
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:
        Follow Policy
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $\boxed{V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]}$ Update values based on TD rule
        $S \leftarrow S'$
    until $S$ is terminal




- **The value estimate for the state node at the top of the backup diagram is updated on the basis of the one sample transition from it to the immediately following state.**

# TD Prediction

## TD *error*

- **TD error** arises in various forms through-out reinforcement learning

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

- **This *error* measures the difference between the estimated value of state [$V(S_t)$] and the better estimate [$R_{t+1} + \gamma V(S_{t+1})$]**

  - The TD error at each time is the error in the estimate ***made at that time***.
  - Because the TD error at step t depends on the next state and next reward, it is not actually available until step t + 1.
  - Updating the value function with the TD error is called a backup. The TD error is related to the Bellman equation.

# TD Prediction

## Summary

**Temporal Difference**

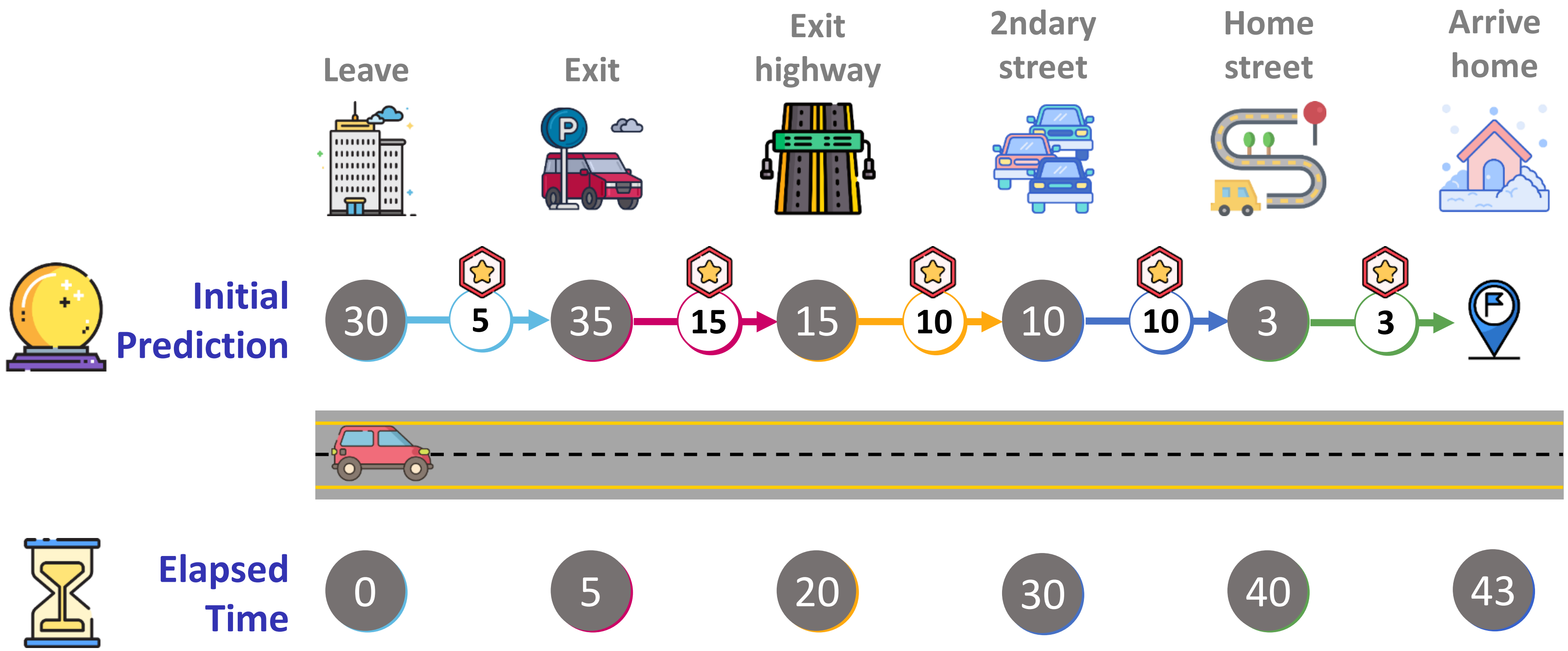
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- **TD updates the value of one state towards its own estimate of the value in the next state.**
- **Value of the next state as a *stand-in* for the return until the end of the episode.**

- **Gamma ($\gamma$): the discount rate**
  - The higher the value the less you are discounting.
- **Delta ($\delta$): a change or difference in value**
  - Measuring the difference between the estimated value of state $[V(S_t)]$ and the better estimate $[R_{t+1} + \gamma V(S_{t+1})]$
- **Alpha ($\alpha$): step size or the learning rate**
  - How much of the error should we accept and therefore adjust our estimates towards.

# TD Prediction

## Example: Driving Home (1)



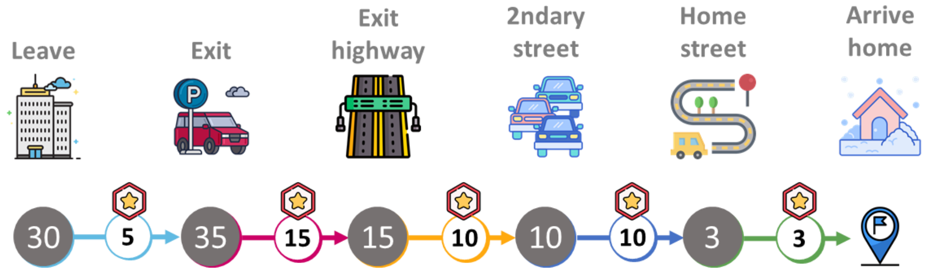
## Example: Driving Home (2)

**Monte Carlo Methods**

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

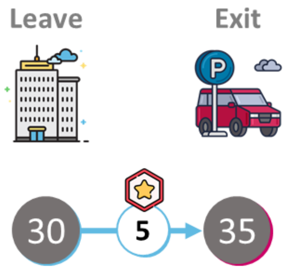
**Temporal Difference**

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




$$G_0 = 5 + 15 + 10 + 10 + 3 = \mathbf{43}$$

$$V(leave) \leftarrow 30 + 1 \times (43 - 30) = 43$$


$$V(leave) \leftarrow V(leave) + \alpha[R_1 + \gamma V(exit) - V(leave)]$$

$$V(leave) \leftarrow 30 + 1 \times [5 + 1 \times 35 - 30] = 40$$

# TD Prediction

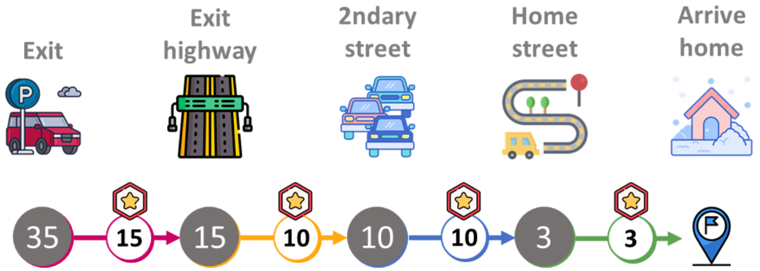
## Example: Driving Home (3)

**Monte Carlo Methods**

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

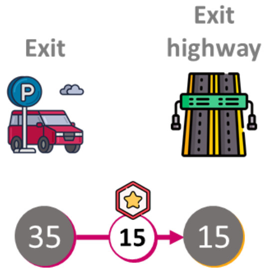
**Temporal Difference**

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




$$G_1 = 15 + 10 + 10 + 3 = 38$$

$$V(leave) \leftarrow 35 + 1 \times (38 - 35) = 38$$


$$V(exit) \leftarrow V(exit) + \alpha[R_1 + \gamma V(highway) - V(exit)]$$

$$V(exit) \leftarrow 35 + 1 \times [15 + 1 \times 15 - 35] = 30$$

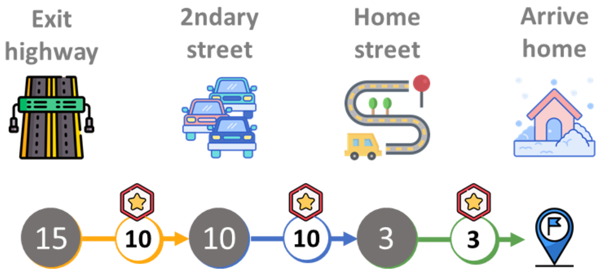
## Example: Driving Home (4)

### Monte Carlo Methods

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

### Temporal Difference

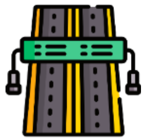
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$



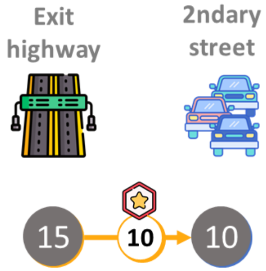

$$G_2 = 10 + 10 + 3 = \mathbf{23}$$

$$V(Hway) \leftarrow 15 + 1 \times (23 - 15) = 23$$

Exit highway



$V(\mathrm{Hway})$




$$V(Hway) \leftarrow V(Hway) + \alpha\left[R_1 + \gamma V\left(2^{nd} St\right) - V(Hway)\right]$$

$$V(Hway) \leftarrow 15 + 1 \times [10 + 1 \times 10 - 15] = 20$$

# TD Prediction

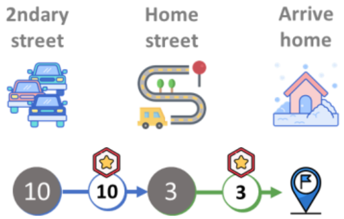
## Example: Driving Home (5)

**Monte Carlo Methods**

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

**Temporal Difference**

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




$$G_3 = 10 + 3 = \mathbf{13}$$

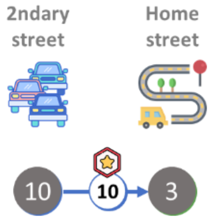
$$V(2^{nd}St) \leftarrow 10 + (13 - 10) = 13$$


$$G_4 = 3 = \mathbf{3}$$

$$V(HomeSt) \leftarrow 3 + (3 - 3) = 3$$

$2^{nd}St$

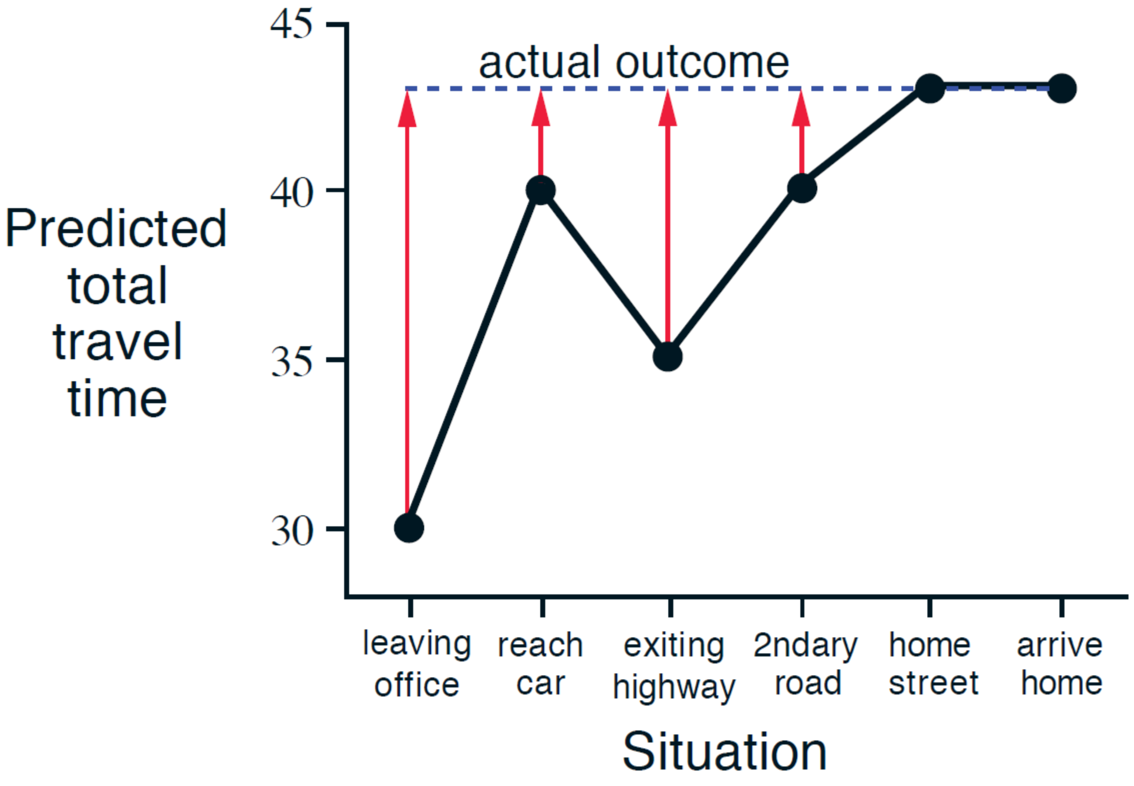
$HomeSt$



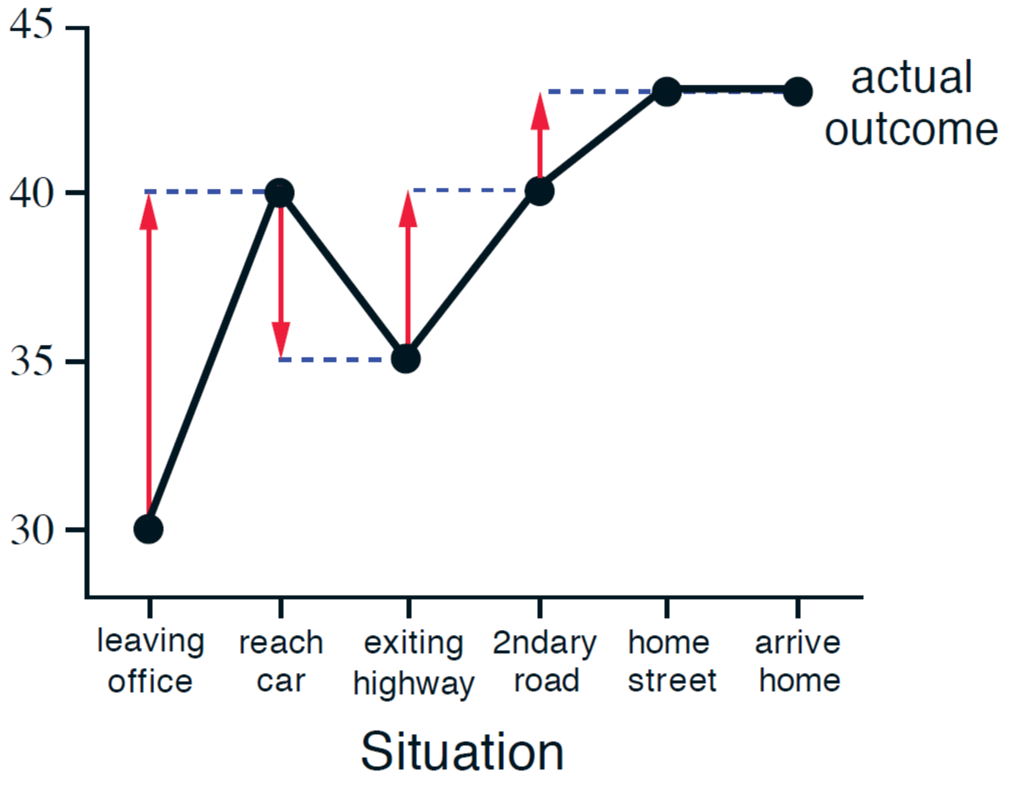

$$V(2^{nd}St) \leftarrow 10 + [10 + 3 - 10] = 13$$


$$V(HomeSt) \leftarrow 3 + [3 + 0 - 3] = 3$$

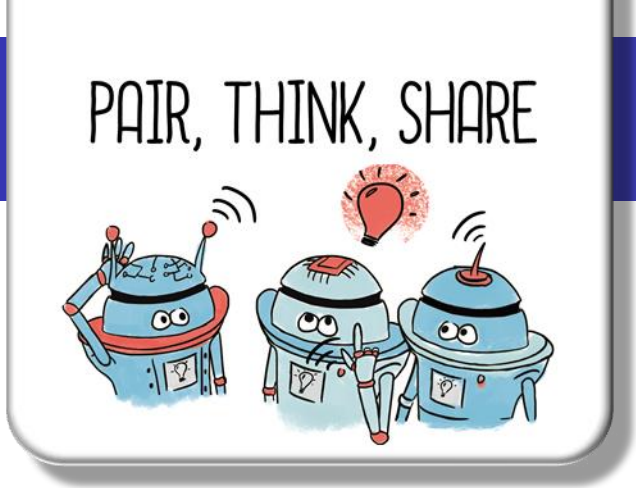
## Example: Driving Home (6)



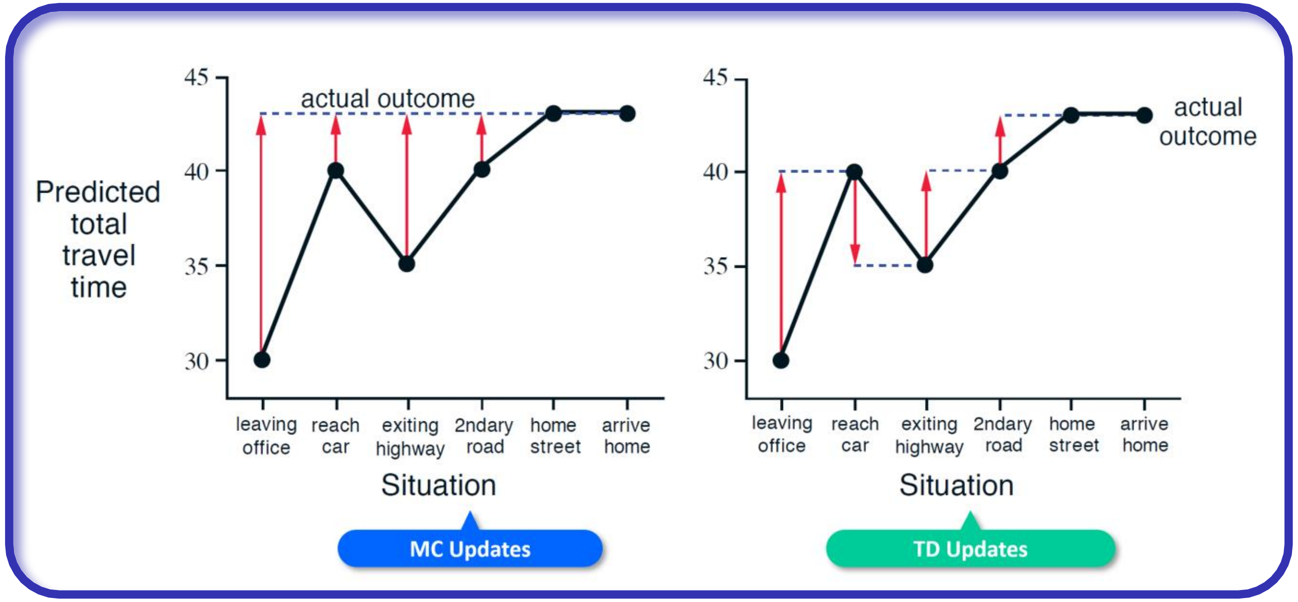

MC Updates




TD Updates




- **Driving home example:** Suppose there is an environment with 2 states
  - 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.
  - **Hint:** Suppose you have lots of experience driving home from work. Then you move to a new building and a new parking lot. Can you see why TD updates are likely to be much better, at least initially, in this case?

# Advantages of TD

# Advantages of TD Prediction Methods

## do not require a model of the environment

- **No knowledge of**
  - MDP transitions
  - rewards

## Online and fully incremental

- **You can learn *before* knowing the final outcome**
  - Less memory
  - Less peak computation
- You can learn ***without*** the final outcome
  - From incomplete sequences

## Faster Convergence

**This is proved experimentally, not mathematically yet ☺**

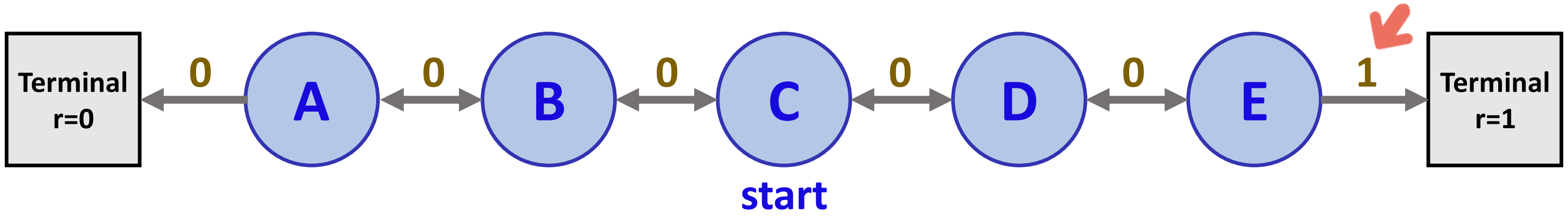
- **TD methods need to wait only one time step**
- **Carlo methods one must wait until the end of an episode**
  - because only then is the return known

## Better Performance

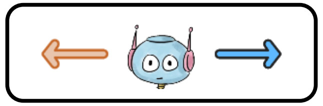
- **TD > MCM in Applications with:**
  - very long episodes
  - continuing tasks
- **Some MCMs must ignore or discount episodes on which experimental actions are taken**

# Advantages of TD

## Example: Random Walk

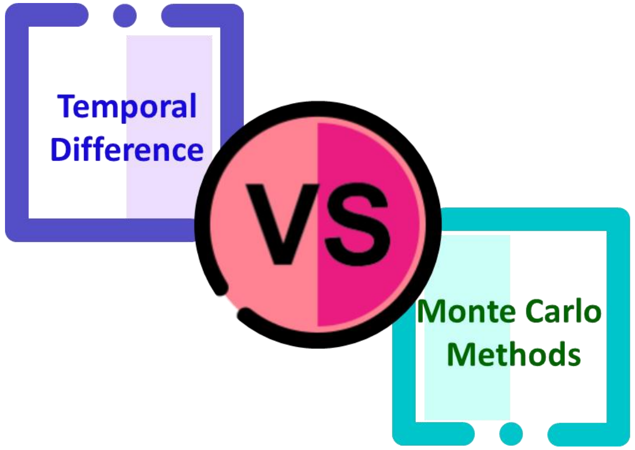



**Actions**

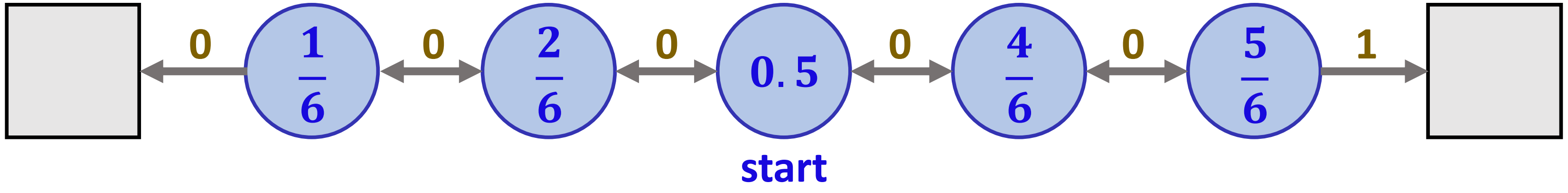


**Policy** $\pi(.|s) = 1/2\,, \forall\, s \in \mathcal{S}$

**Settings** $\gamma = 1$



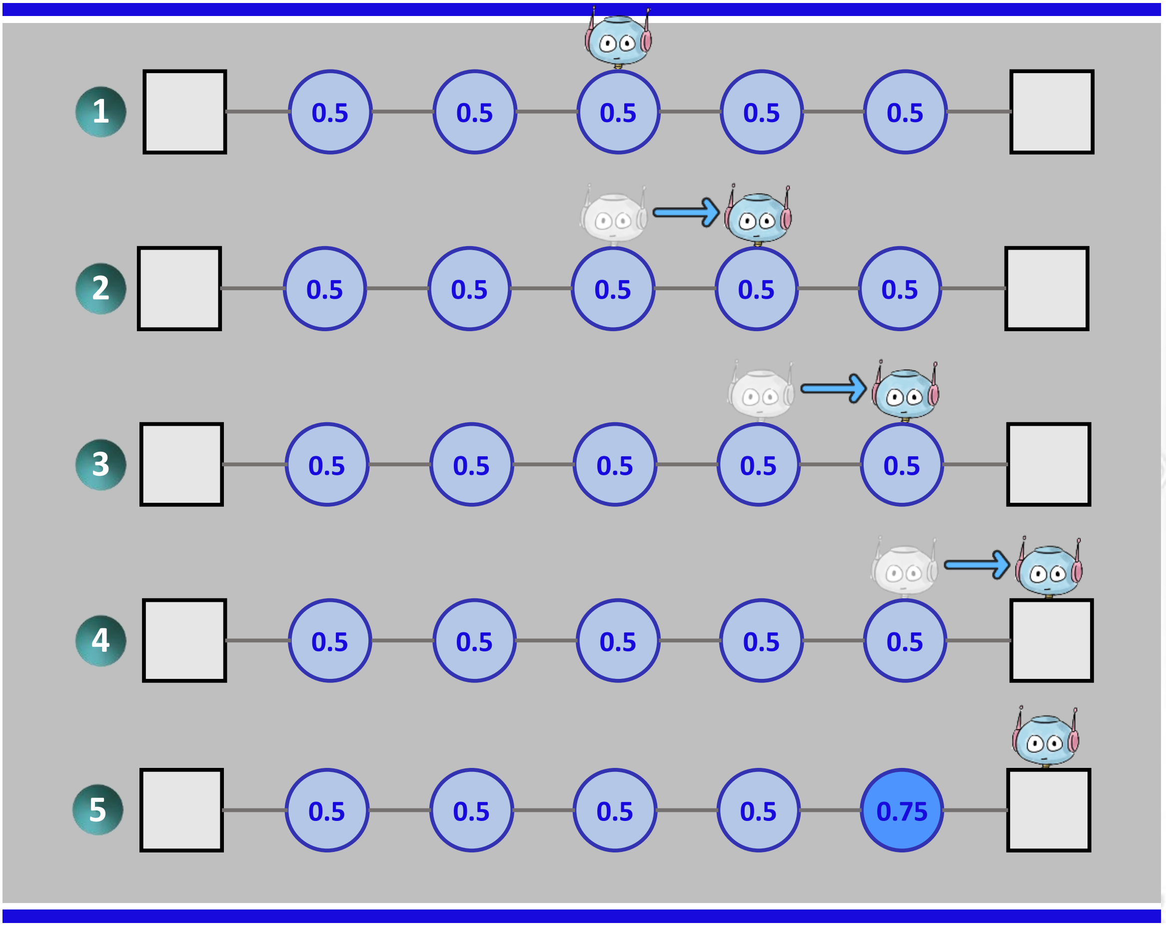

**Target (Exact Values)**



# Advantages of TD

## Example: Random Walk (2)

**Temporal Difference**

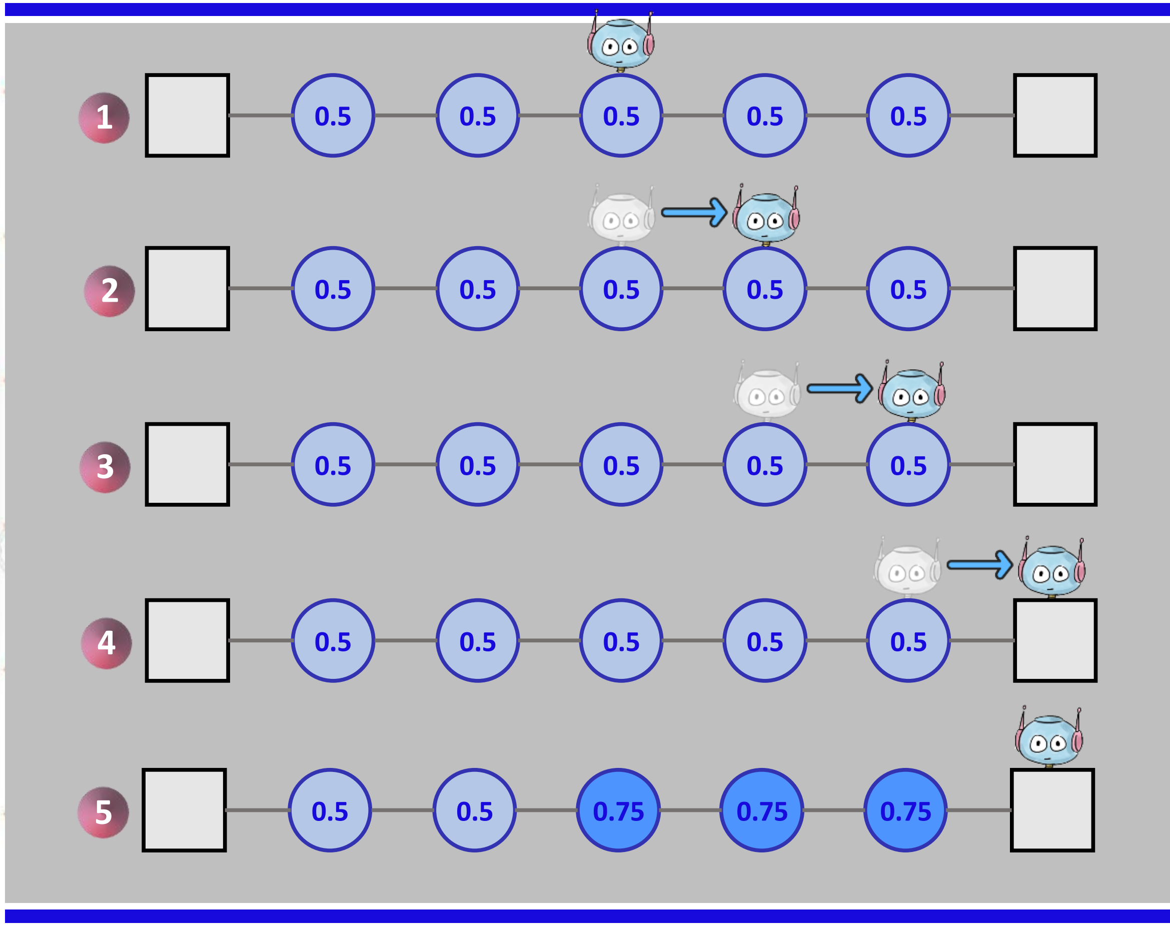



Action:

Action:

Action:

**Monte Carlo Methods**

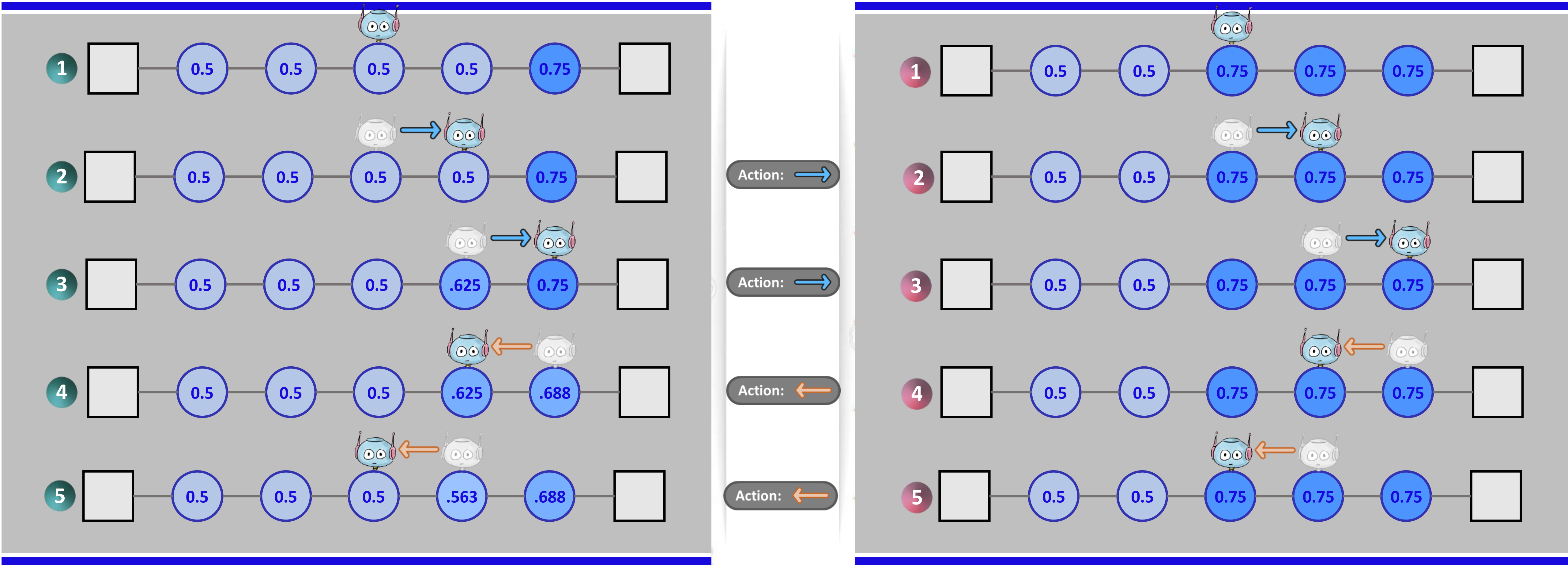


# Advantages of TD

## Example: Random Walk (3)

### Temporal Difference

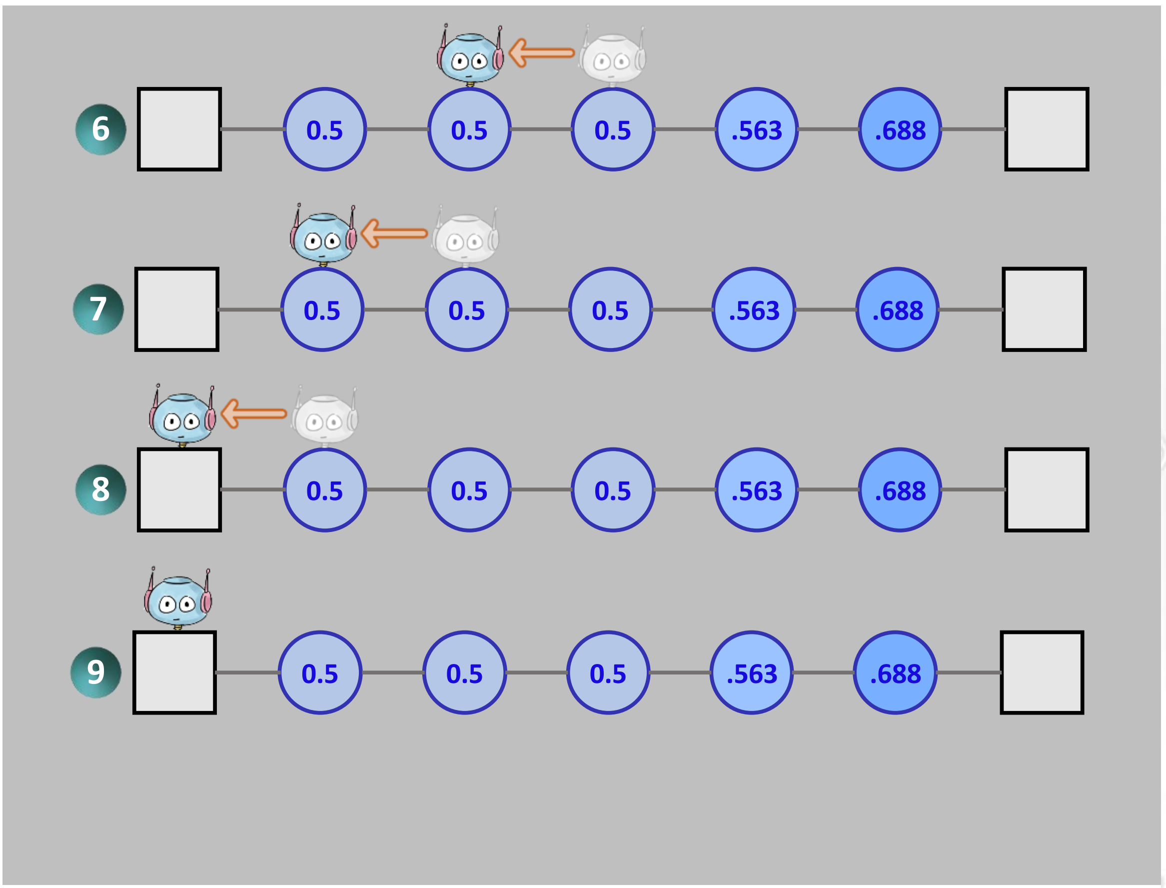
### Monte Carlo Methods



# Advantages of TD

## Example: Random Walk (4)

### Temporal Difference

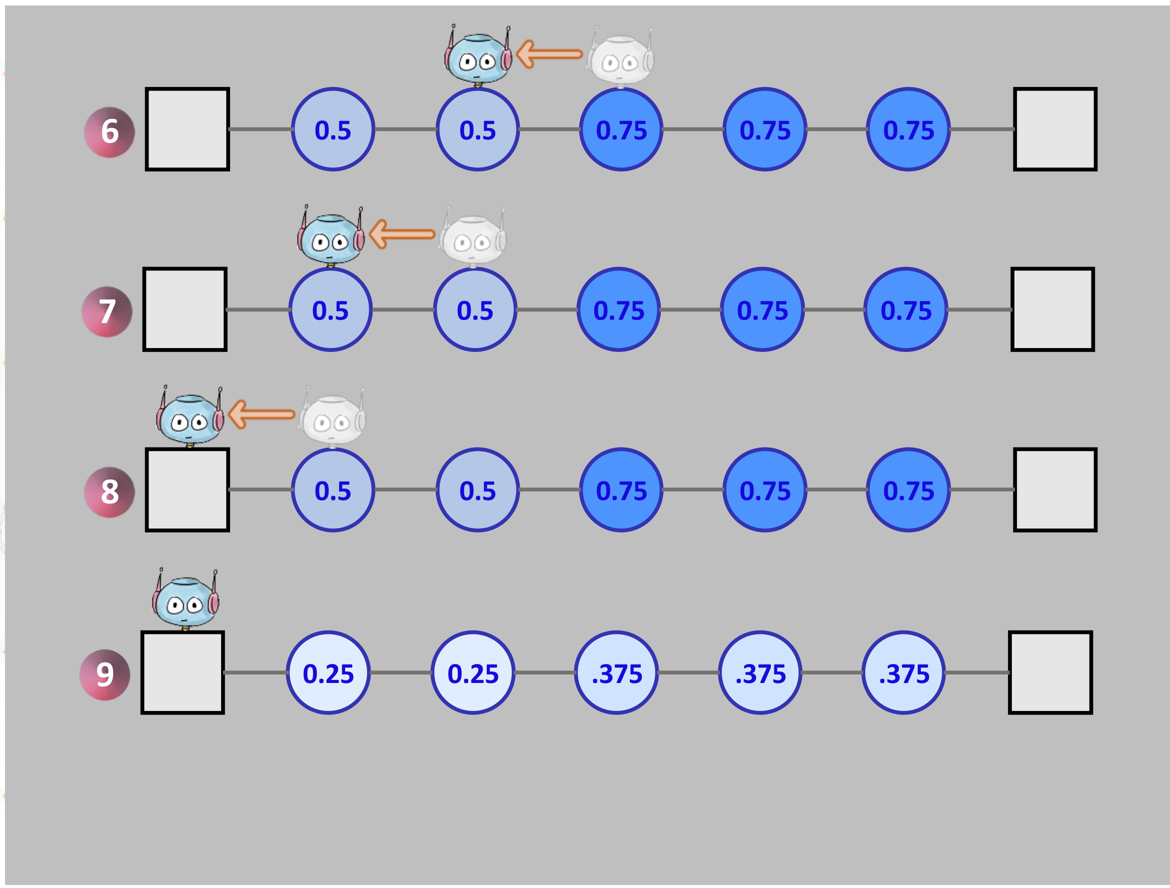



Action: ←

Action: ←

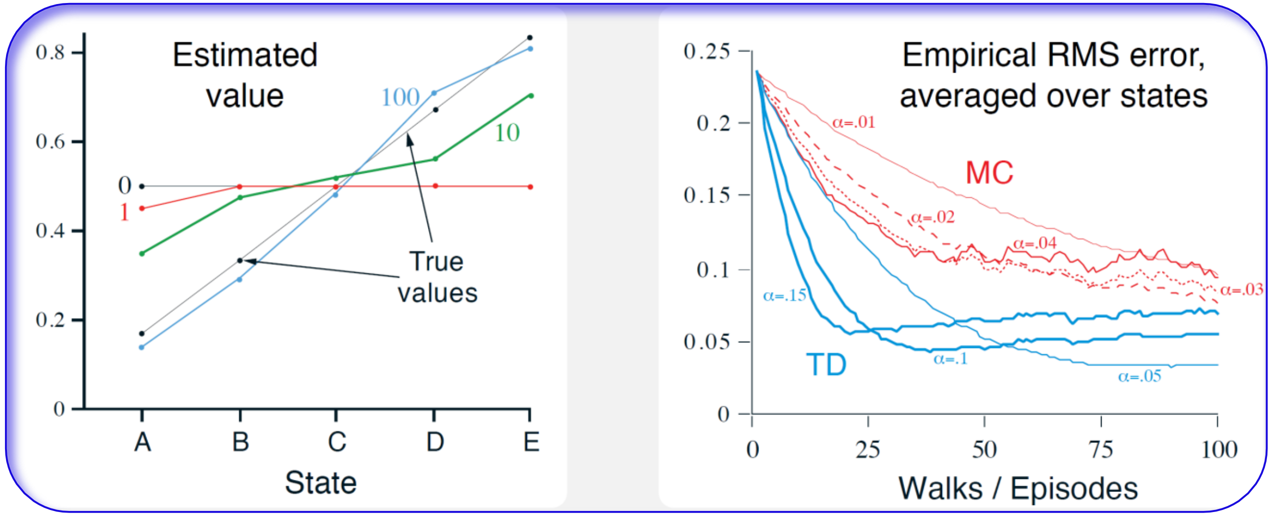
Action: ←

### Monte Carlo Methods



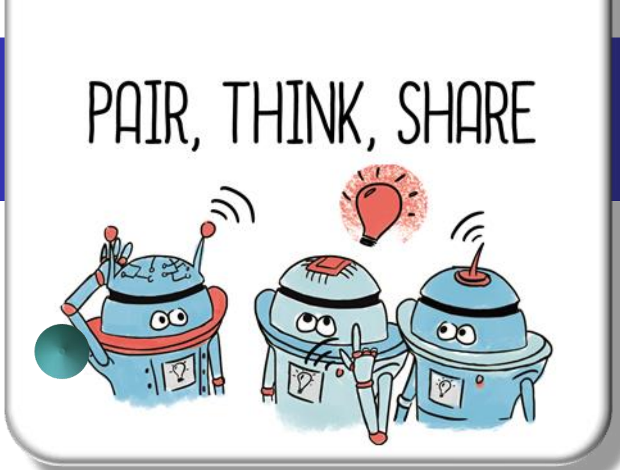
# Advantages of TD

## Results: Random Walk




- **Experiment:**
  - $V(S) = 0.5$
  - 100 episodes
    - **Left Graph:** single run of TD(0)
    - **Right graph:** results are averaged over 100 runs

- **The TD method was consistently better than the MC method on this task**
- **The estimates after 100 episodes are about as close as they ever come to the true values**

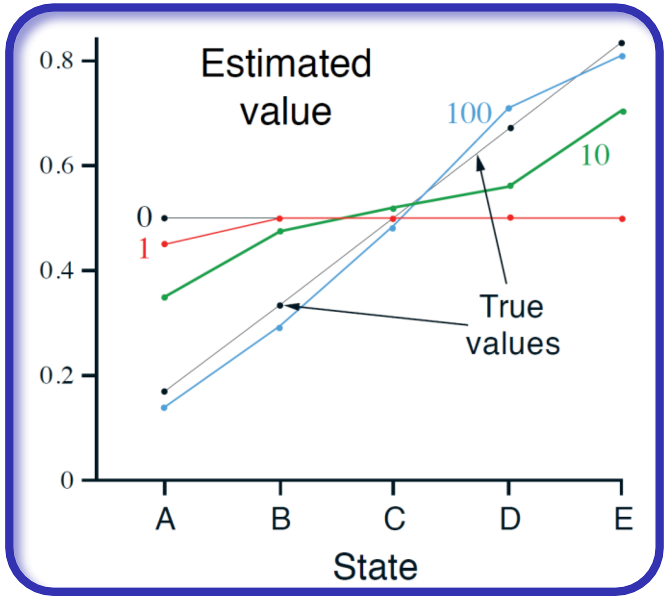



## Random Walk Example-a:

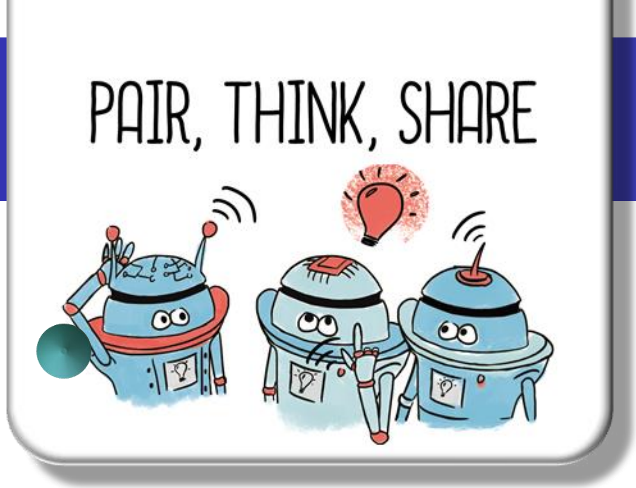
Consider the results in the graph (a). It appears that the first episode results in a change in only $V(A)$.

## 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

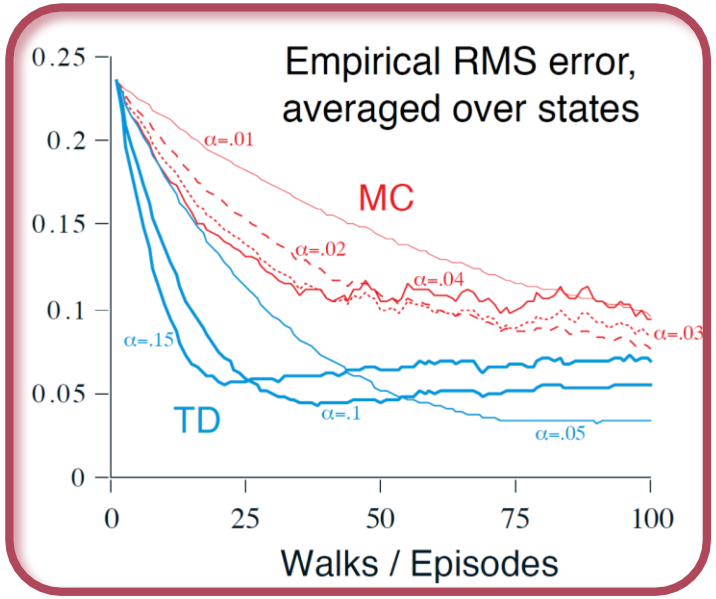



(a)




## Random Walk Example-b:

Consider the results in the graph (b). It appears that the results are dependent on the value of the step-size parameter, $\alpha$.




**(b)**

## Do you think the conclusions about which algorithm is better would be affected if a wider range of $\alpha$ values were used?

- Is there a different, fixed value of $\alpha$, at which either algorithm would have performed significantly better than shown? Why or why not?

# Convergence of TD methods

## Convergence Conditions (Robbins-Monro conditions)

There are two conditions required to assure convergence with probability

1 $$\sum_{n=1}^{\infty} \alpha_n(a) = \infty$$ and 2 $$\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

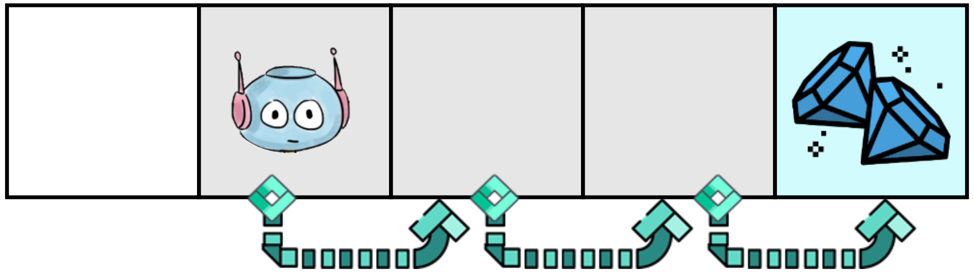
- The first condition is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations.
- The second condition guarantees that eventually the steps become small enough to assure convergence.
- the choice $\alpha_n(a) = \frac{1}{n}$ results in the sample-average method, which is guaranteed to converge to the true action values by the law of large numbers.

# Sarsa:
# On-policy
# TD Control

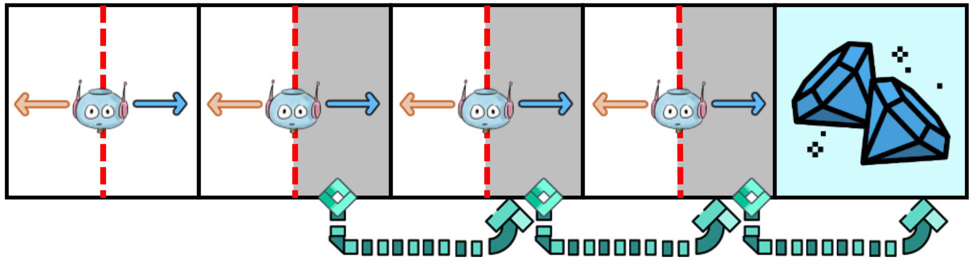
**State to state**

**State-action to state-action**

$V(s)$

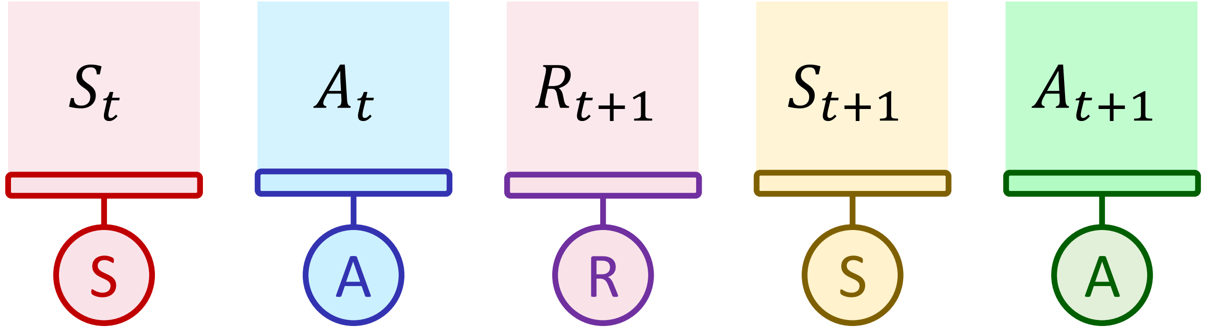


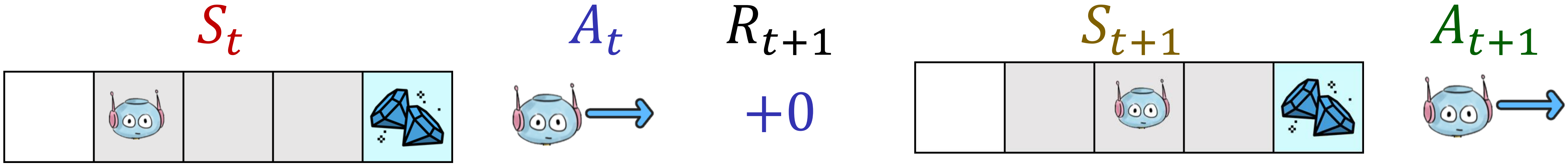
$Q(s,a)$

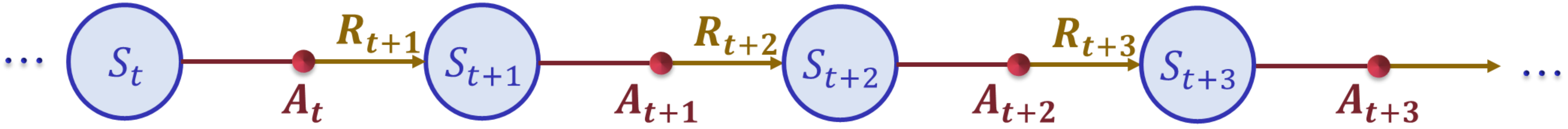


# Sarsa

## Defintion






### Sarsa Update Equation

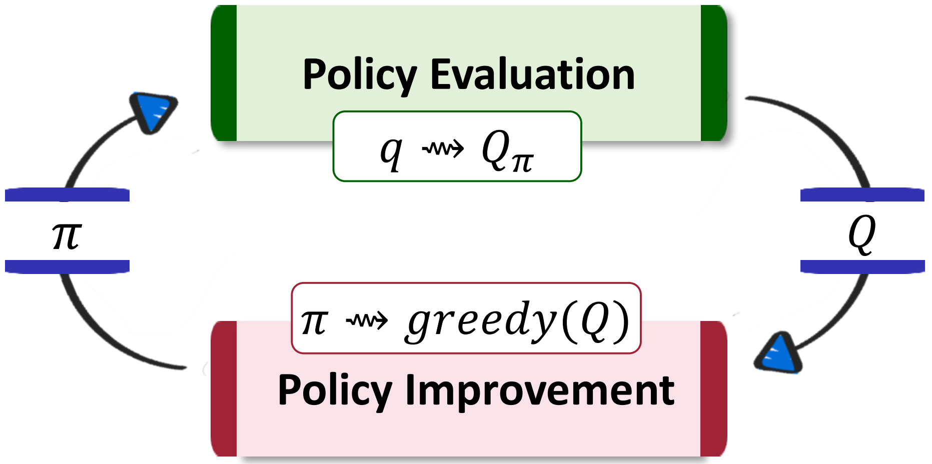
- **The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values**

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

  - This update is done after every transition from a nonterminal state $S_t$

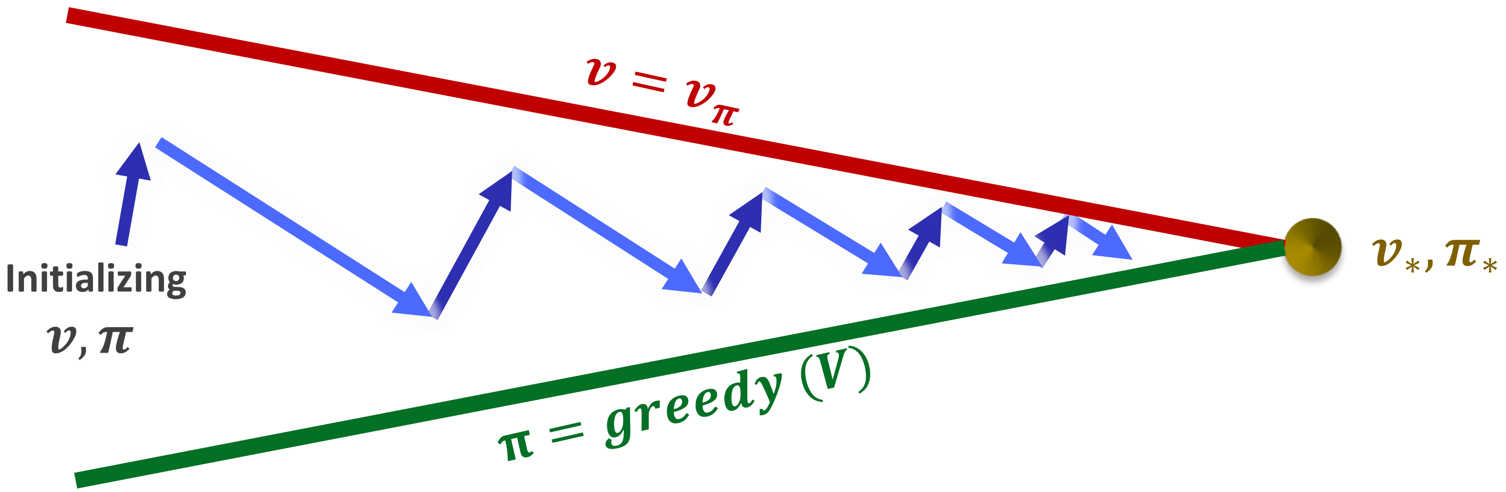
## Control in Sarsa: Generalized policy iteration (GPI)

### How to perform control in Sarsa




### As in all on-policy methods:

- we continually estimate $q_\pi$ for the behavior policy $\pi$,
- at the same time change $\pi$ toward greediness with respect to $q_\pi$



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

Step Size

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 $Q(terminal, \cdot) = 0$

i.e. Optimistic Initial Values

Loop for each episode:

  Initialize $S$

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

Stochastic policy

  Loop for each step of episode:

Main Loop

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

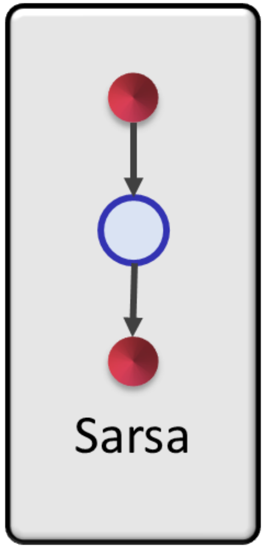
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)

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

Sarsa Update Equation

    $S \leftarrow S'$; $A \leftarrow A'$;

  until $S$ is terminal



# Q-Learning: Off-policy TD Control

# Q-Learning

## Introduction

- **One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989)**
  - Just a few years after the emergence of TD in 1984

- **Q-learning can be viewed as a method of asynchronous dynamic programming (DP)**
  - It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains.

- **Q-learning uses an off-policy control that separates the deferral policy from the learning policy**
  - It updates the action selection using the Bellman optimal equations and the e-greed policy

- **Since the emergence of Q-learning, many studies have described its uses in reinforcement learning and artificial intelligence problems.**

### Early days applications

- **process control**
  - chemical process
  - industrial process automatic control
  - airplane control

### Modern Applications

- **network management**
  - Routing
  - network communication
- **Gaming:**
  - AlphaGo

# Q-Learning

## Update equation

- **Q-learning has simple Q-functions, hence it has become the foundation of many other reinforcement learning algorithms**

### Q-learning Update Equation

- **In Q-learning, the learned action-value function, Q, directly approximates $q_*$, the optimal action-value function**

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

Off-Policy

- This update independent of the policy being followed (off-Policy)

## Sarsa

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

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

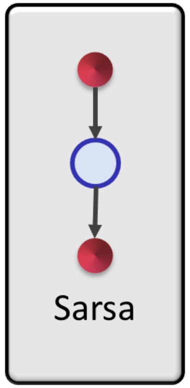
## Q-learning

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

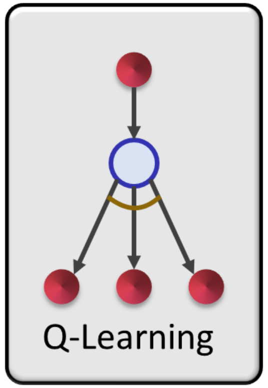
$$q_*(s, a) = \sum_{s', r} p(s', r | s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]$$

## Sarsa vs Q-learning (2)

- **Sarsa is sample-based version of *policy iteration***
  - It uses Bellman equations for action values, that each depend on a fixed policy.
  - SARSA uses the estimate of the next action value in its target. This changes every time the agent takes an exploratory action.




- **Q-learning is a sample-based version of *value iteration* which iteratively applies the Bellman optimality equation.**
  - Applying the Bellman's Optimality Equation strictly improves the value function, unless it is already optimal.
  - Q-Learning takes the max over next action values. So it only changes when the agent learns that one action is better than another.



# Q-Learning

## Pseudocode: Q-learning control

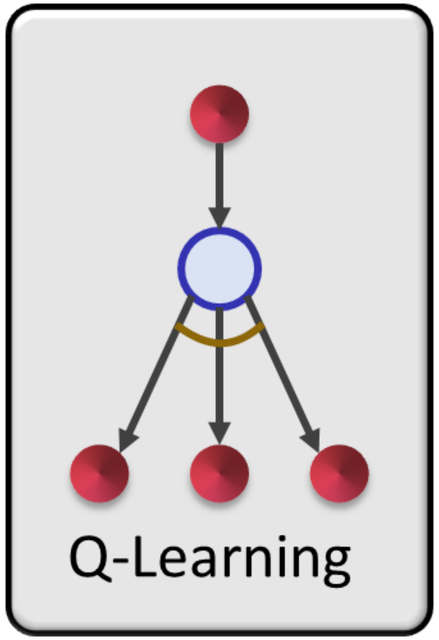
**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

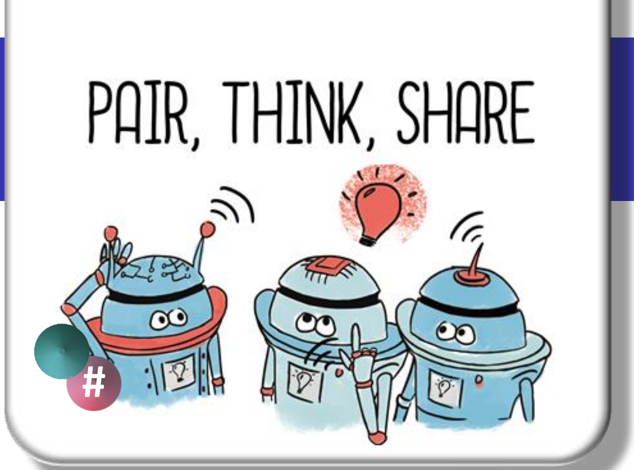
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

**i.e. Optimistic Initial Values**

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

**Q-learning Update Equation**





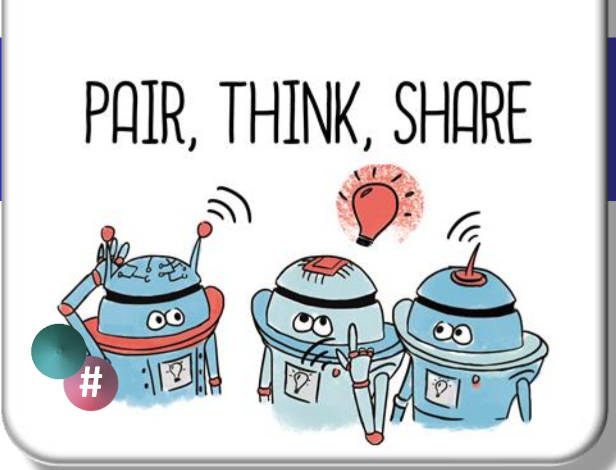

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Off-Policy

- **Why is Q-learning considered an off-policy control method?**

- **Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates?**

- **If Q-learning learns off-policy, why don't we see any important sampling ratios?**




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

Off-Policy

## Why is Q-learning considered an off-policy control method?

Q-learning learns about the greedy policy, which gradually becomes the optimal policy, independent of what policy is actually being followed by the agent (as long as it tries all state-action pairs). Thus, it is learning about one policy while following another, which by definition makes it an o-policy method.

## Suppose action selection is greedy. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates?

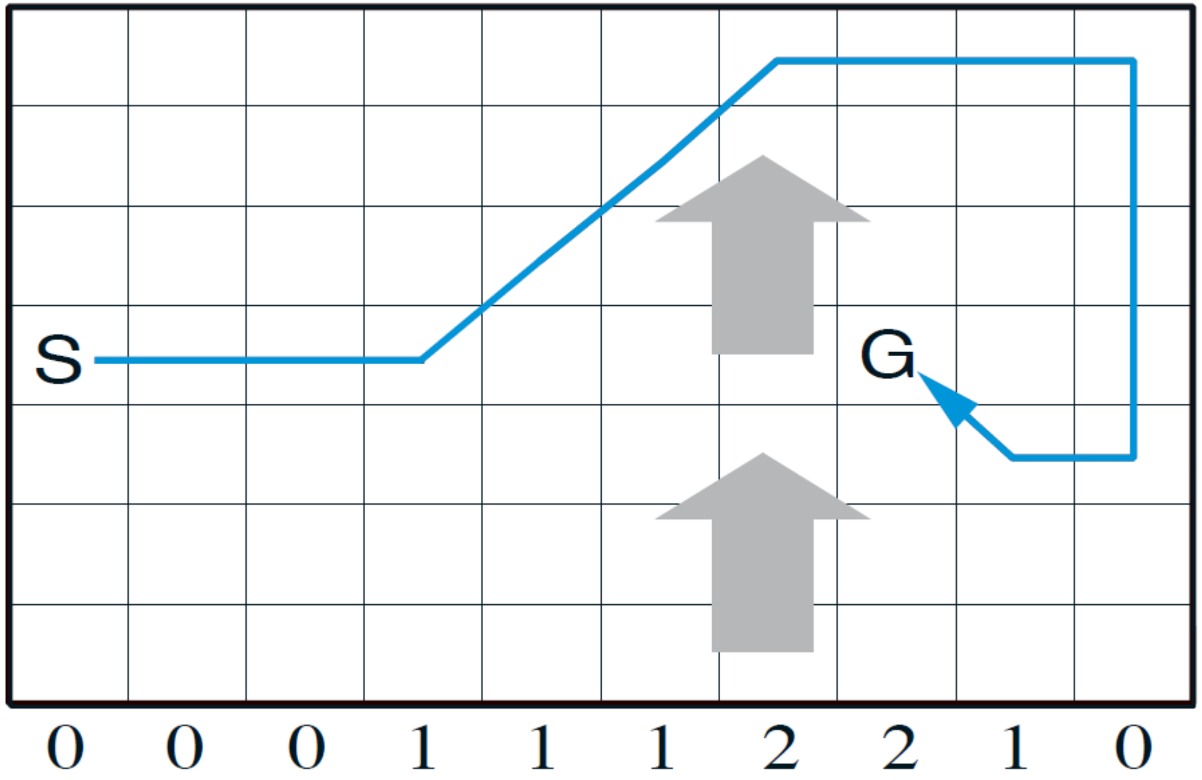
The two algorithms will actually be subtly different. In particular, they will not always make the same action selections. To see this, consider a transition t→ t+1 in which the state does not change and the learning update changes which action is the greedy action. On time step t+1, Q-learning will take the newly greedy action, but Sarsa will take the formerly greedy action. This is because Sarsa must commit to At+1 at t, before the update, as this action is used in the update at time t.

## If Q-learning learns off-policy, why don't we see any important sampling ratios?

Since the agents target policies greedy, with respect to its action values, all non-maximum actions have probability 0. As a result, the expected return from that state is equal to a maximal action value from that state.

# Q-Learning and Sarsa

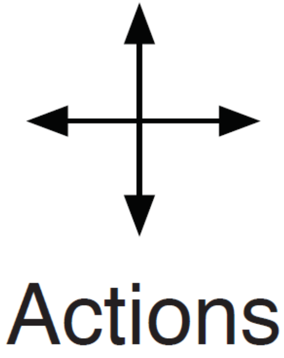
## Lab: Windy Gridworld




$$\varepsilon = 0.1$$

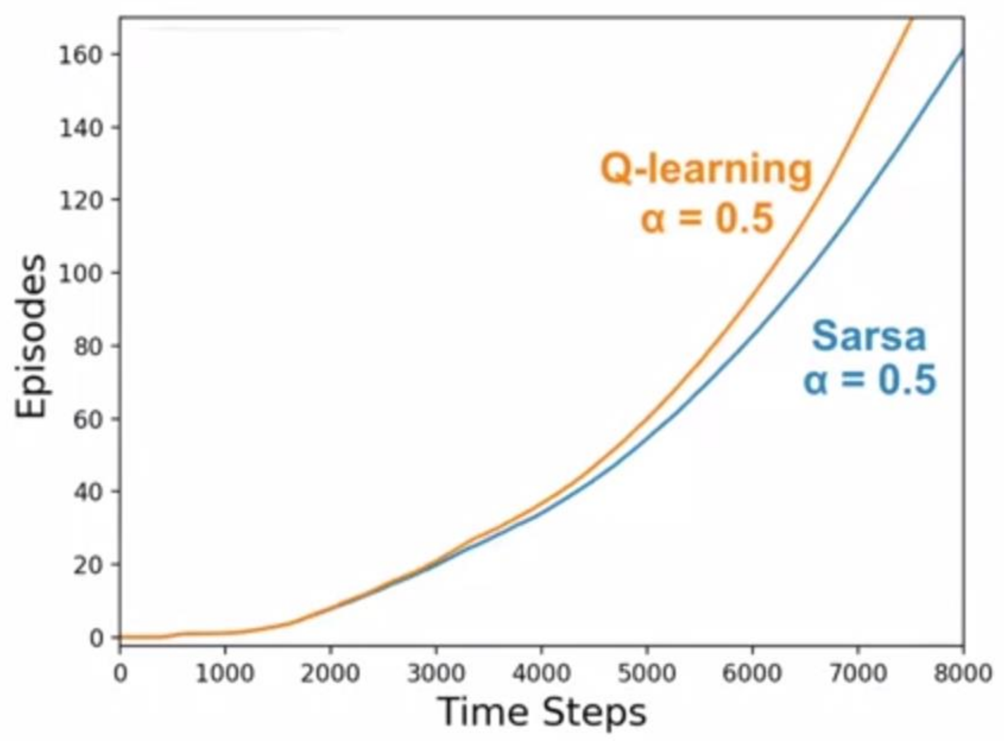
$$\alpha = 0.5$$

$$\text{initial values } Q(s,a) = 0$$



# Q-Learning and Sarsa

## Lab: Windy Gridworld





python

# Expected Sarsa

### Expected Sarsa Update Equation

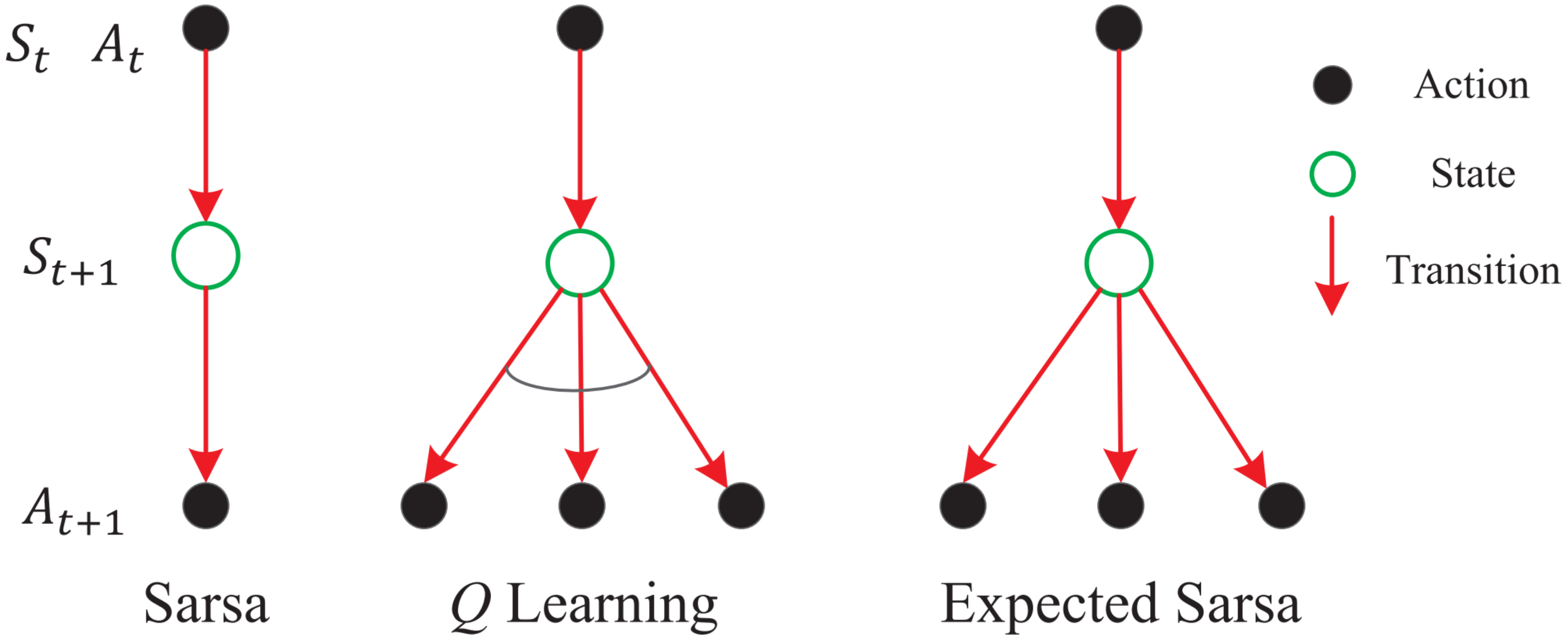
- Consider the learning algorithm that is just like Q-learning except that instead of the maximum over next state–action pairs it uses the expected value, taking into account how likely each action is under the current policy.

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

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

- **Given the next state, St+1, this algorithm moves deterministically in the same direction as Sarsa moves in expectation, and accordingly it is called Expected Sarsa.**

# Summary



$S_t$ $A_t$
$S_{t+1}$
$A_{t+1}$
Sarsa
$Q$ Learning
Expected Sarsa
Action
State
Transition

# References


- [1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.
- [2] “Simple Reinforcement Learning: Temporal Difference Learning.” [Online]. Available: https://medium.com/@violante.andre/simple-reinforcement-learning-temporal-difference-learning-e883ea0d65b0. [Accessed: 22-Feb-2020].
- [3] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, “Q-learning algorithms: A comprehensive classification and applications,” IEEE Access, vol. 7, pp. 133653–133667, 2019.
- [4] P. Dayan and C. Watkins, “Q-learning,” Machine learning, vol. 8, no. 3, pp. 279–292, 1992.
- [5] H. Jiang, R. Gui, Z. Chen, L. Wu, J. Dang, and J. Zhou, “An Improved Sarsa ($$\backslash$lambda $) Reinforcement Learning Algorithm for Wireless Communication Systems,” IEEE Access, vol. 7, pp. 115418–115427, 2019.