ELTE | FACULTY OF INFORMATICS

TEMPORAL DIFFERENCE

Deep Reinforcement Learning

Balázs Nagy, PhD

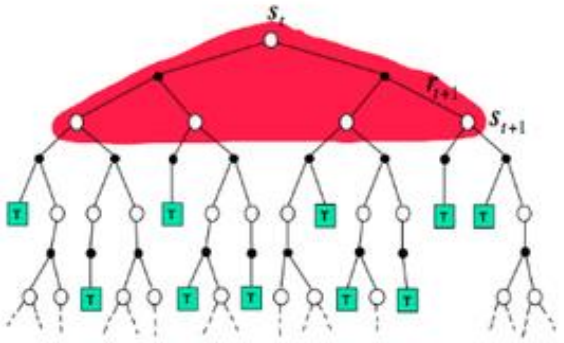
ELTE | IK
DEPARTMENT OF ARTIFICIAL INTELLIGENCE

RL ideas compared

DP	MC	TD
Model of the environment needed	**No Model needed** **Learn from raw experience**	**No Model needed** **Learn from raw experience**
Use Bootstrap **Update estimates based on other estimates**	Not use Bootstrap Wait for episode outcome	**Use Bootstrap** **Update estimates based on other estimates**
Expected update	**Sample update**	**Sample update**

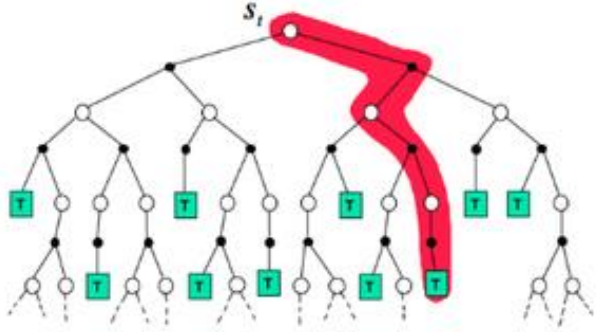
Dynamic Programming

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



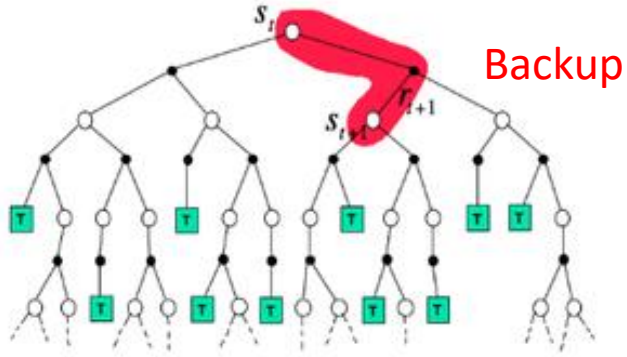
Monte-Carlo

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



Temporal-Difference

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



Backup diagrams

TD prediction

- Simple every-visit Monte Carlo method suitable for nonstationary environments

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

MC methods must wait until the end of the episode to determine the increment

- Simple TD method update

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

TD methods need to wait only until the next time step

Pseudocode

Tabular TD(0) for estimating v_π

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

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

TD error

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

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

The TD error at each time is the error in the estimate made at that time

Driving home example

Each day as you drive home from work, you try to predict how long it will take to get home. When you leave your office, you note the time, the day of week, the weather, and anything else that might be relevant. Say on this Friday you are leaving at exactly 6 o'clock, and you estimate that it will take 30 minutes to get home. As you reach your car it is 6:05, and you notice it is starting to rain. Traffic is often slower in the rain, so you reestimate that it will take 35 minutes from then, or a total of 40 minutes. Fifteen minutes later you have completed the highway portion of your journey in good time. As you exit onto a secondary road you cut your estimate of total travel time to 35 minutes. Unfortunately, at this point you get stuck behind a slow truck, and the road is too narrow to pass. You end up having to follow the truck until you turn onto the side street where you live at 6:40. Three minutes later you are home.

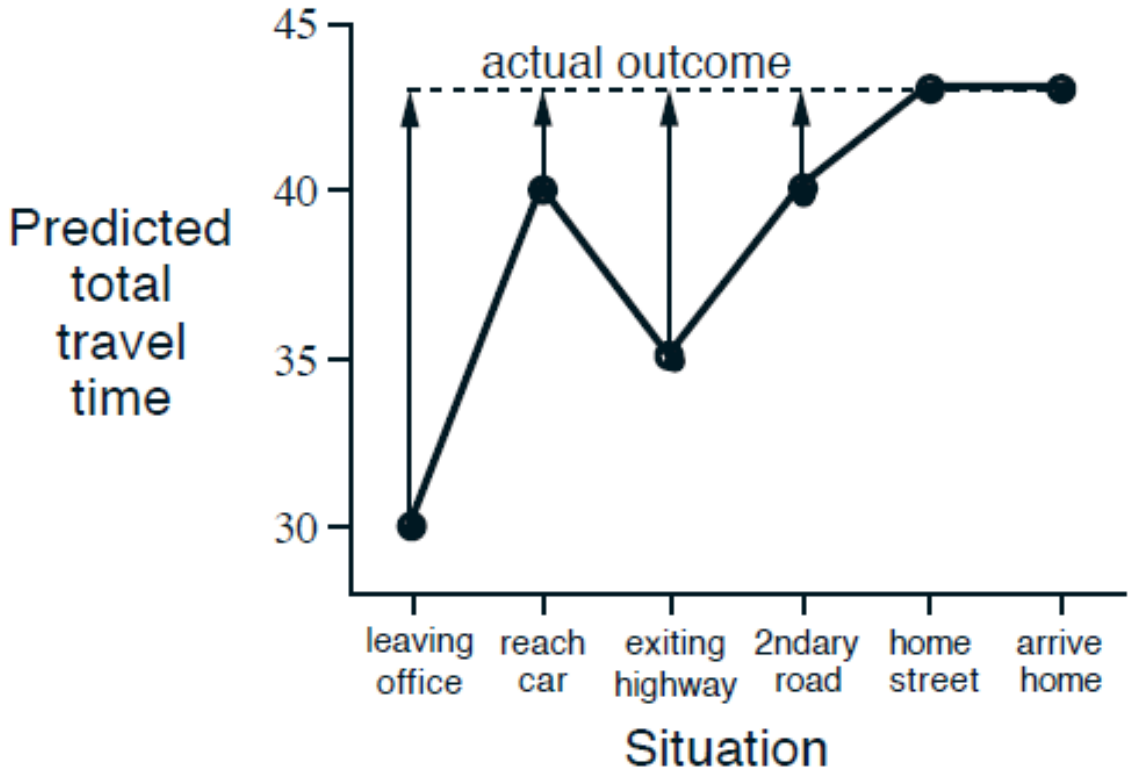
Driving home example

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

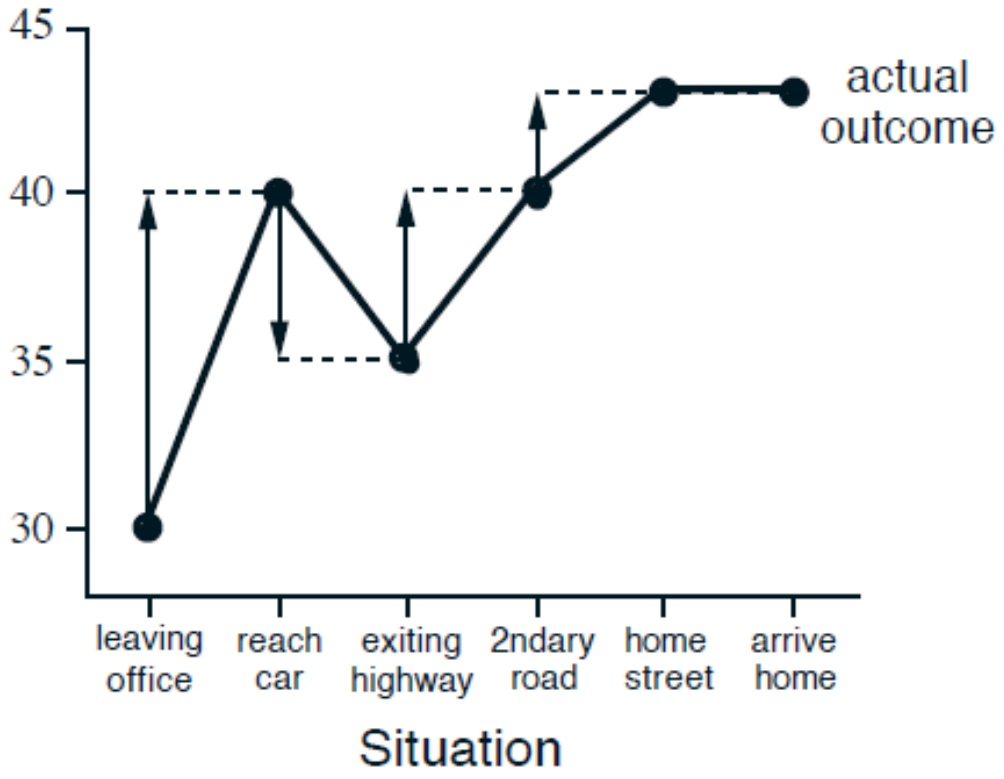
- The rewards are the elapsed times on each leg of the journey
- Not discounting ($\gamma = 1$), and thus the return for each state is the actual time to go from that state
- The value of each state is the expected time to go. The second column of numbers gives the current estimated value for each state encoutered.

Driving home example

MC



TD



Advantages of TD

- No need for a model
- Naturally implemented in an on-line
- Fully incremental (no need to wait for the episode to end)
- No problem with experimental actions

Advantages of TD

- No need for a model
- Naturally implemented in an on-line
- Fully incremental (no need to wait for the episode to end)
- No problem with experimental actions

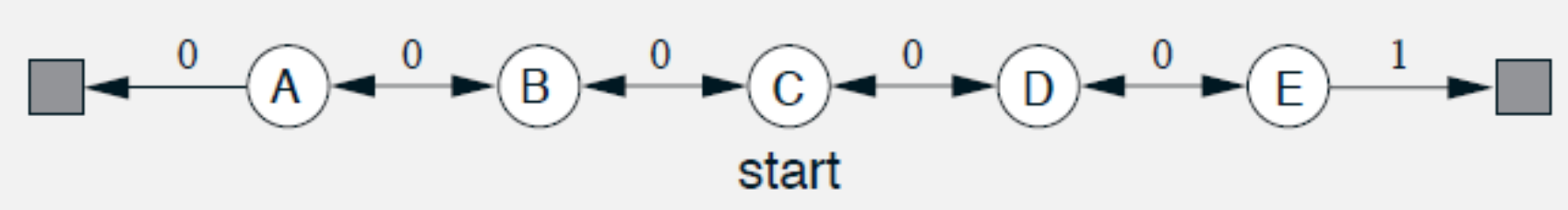
- Faster than MC?
  - No mathematical proof, but…

Random walk example

- A Markov reward process, or MRP, is a Markov decision process without actions
- Focusing on the prediction problem, in which there is no need to distinguish the dynamics due to the environment

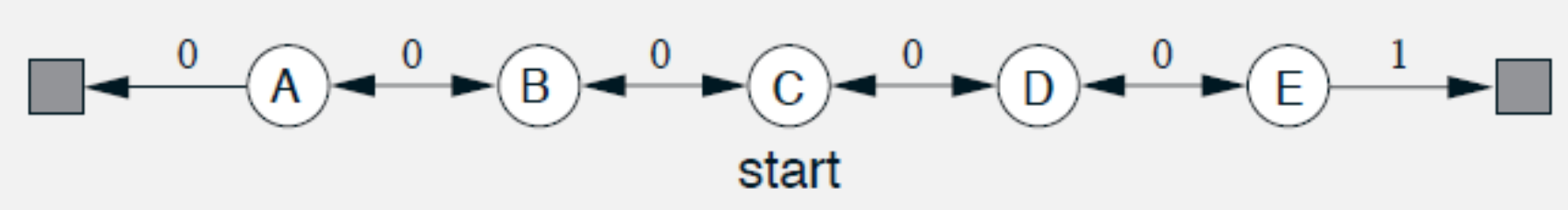
Random walk example

- A Markov reward process, or MRP, is a Markov decision process without actions
- Focusing on the prediction problem, in which there is no need to distinguish the dynamics due to the environment



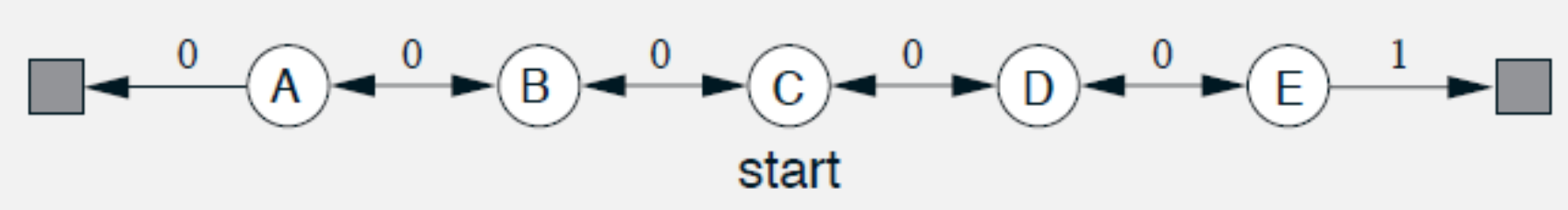
All episodes start in the center state, *C*, then proceed either left or right by one state on each step, with equal probability. Episodes terminate either on the extreme left or the extreme right. When an episode terminates on the right, a reward of +1 occurs; all other rewards are zero

Random walk example



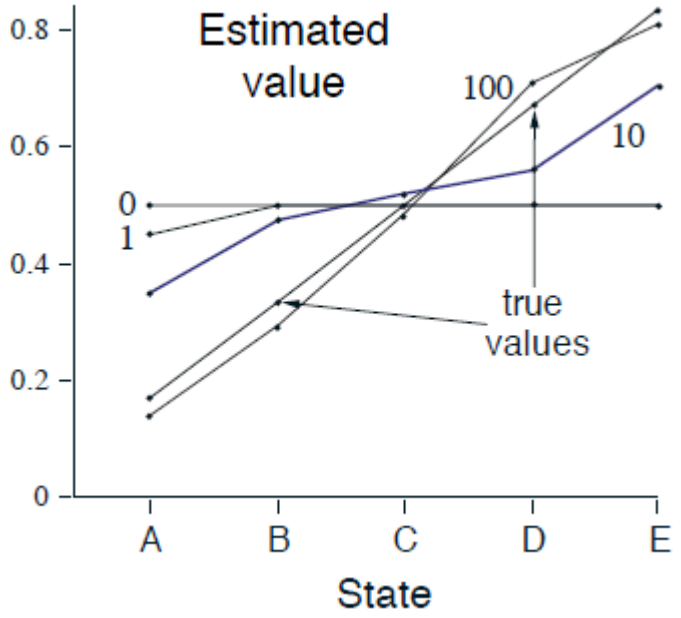
- What are the true state values? (no discounting)

Random walk example

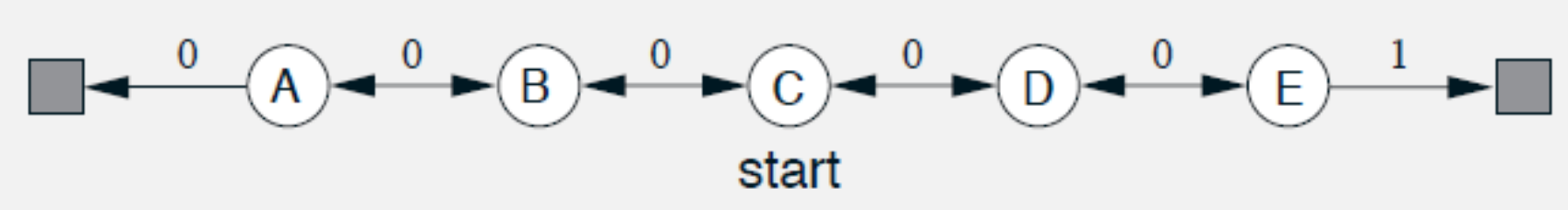


- What are the true state values? (no discounting)

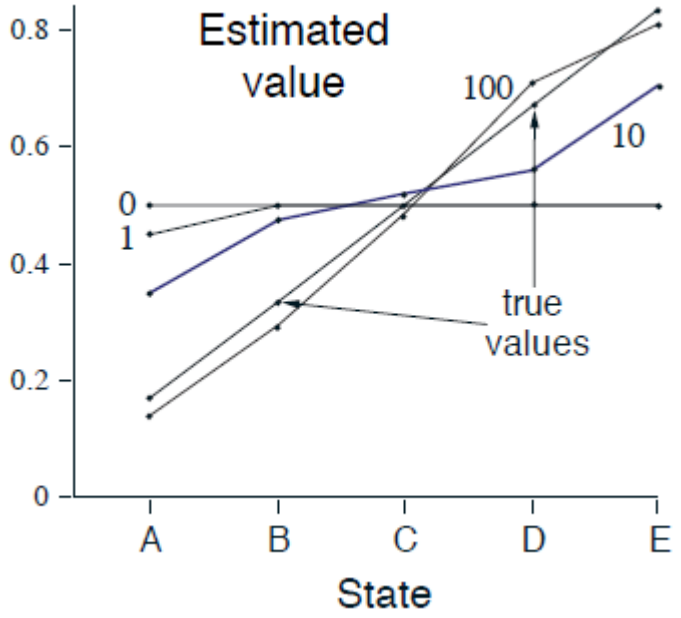
$v_\pi(A) = 1/6$
$v_\pi(B) = 2/6$
$v_\pi(C) = 3/6$
$v_\pi(D) = 4/6$
$v_\pi(E) = 5/6$

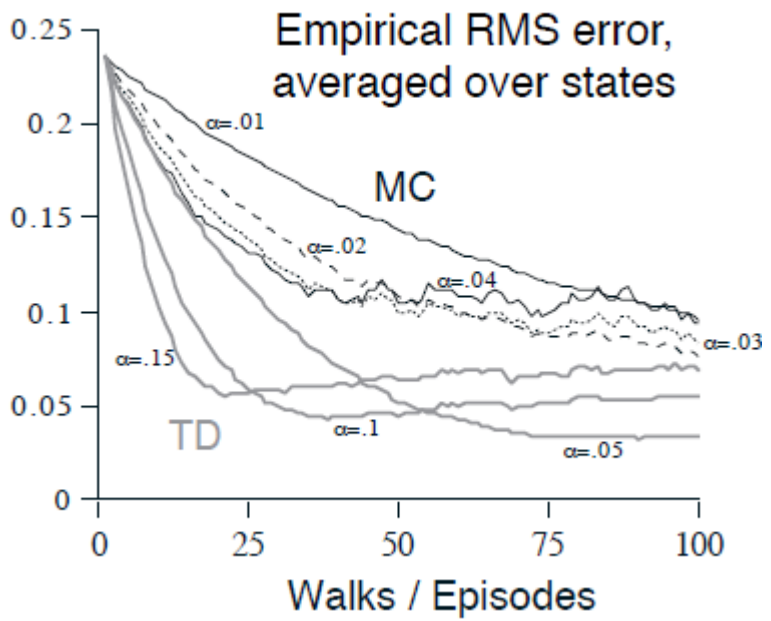


Random walk example



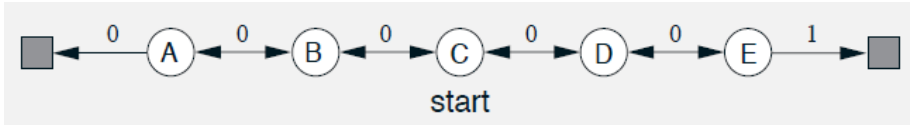
- What are the true state values? (no discounting)
  - $v_\pi(A) = 1/6$
  - $v_\pi(B) = 2/6$
  - $v_\pi(C) = 3/6$
  - $v_\pi(D) = 4/6$
  - $v_\pi(E) = 5/6$

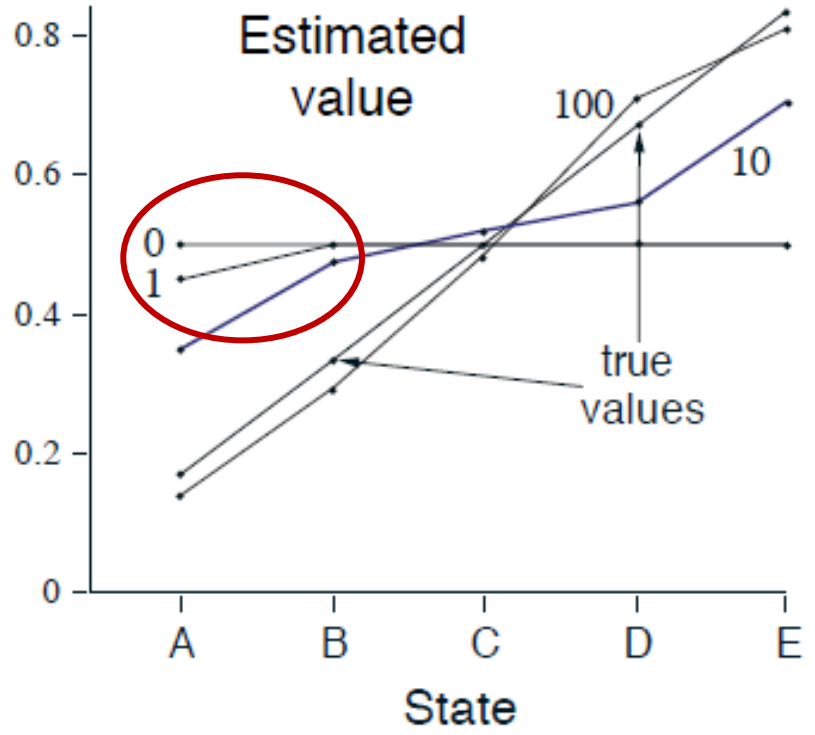




Random walk example

From the results shown in the graph 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 if $\alpha = 0.1$?

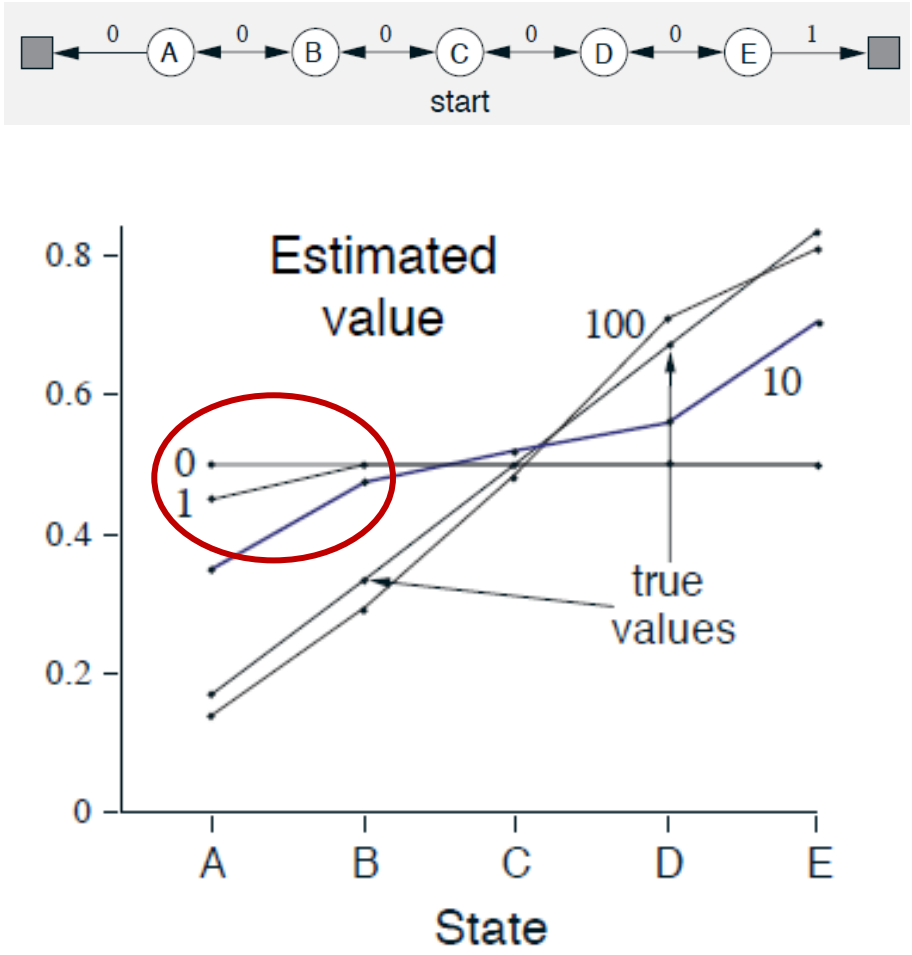




Random walk example

From the results shown in the graph 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 if $\alpha = 0.1$?

$$
\begin{aligned}
V(A) &= V(A) + \alpha\left[R_{t+1} + V(S_T) - V(A)\right] \\
&= 0.5 + 0.1\left[0 + 0 - 0.5\right] \\
&= 0.45
\end{aligned}
$$



Optimality of TD(0)

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

  Compute updates according to TD(0), but only update estimates after each complete pass through the data

Optimality of TD(0)

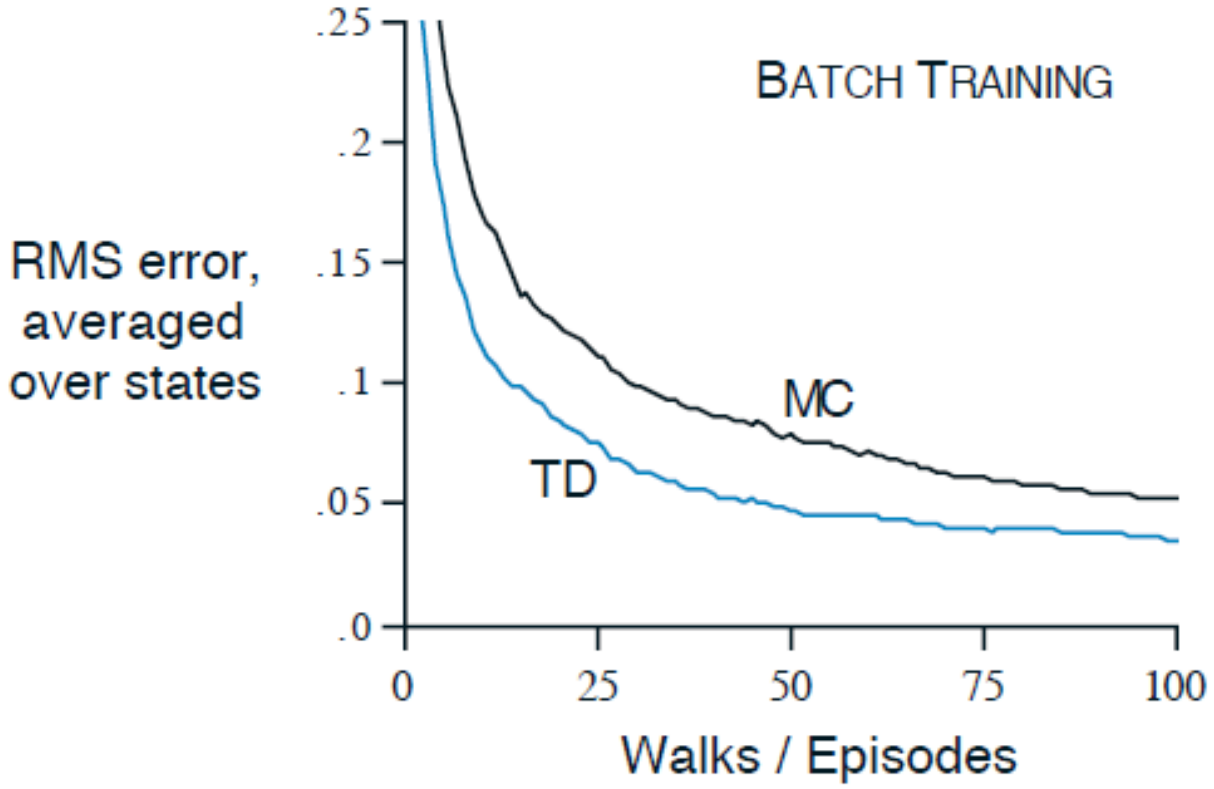
- **Batch Updating**:
  train completely on a finite amount of data
  e.g.: train repeatedly on 10 episodes until convergence

  Compute updates according to TD(0), but only update estimates after each complete pass through the batch

- For any finite Markov prediction task, under batch updating, TD(0) converges for sufficiently small α
- Constant-α MC also converges under these conditions, but to a different answer!

Random walk with batch training

- TD is constantly better



Predictor example

Place yourself in the role of the predictor of returns for an unknown Markov reward process. Suppose you observe the following eight episodes

Episode 1	**A, 0, B, 0**
Episode 2	**B, 1**
Episode 3	**B, 1**
Episode 4	**B, 1**

Episode 5	**B, 0**
Episode 6	**B, 1**
Episode 7	**B, 1**
Episode 8	**B, 1**

Given this batch of data, what would you say are the optimal predictions, the best values for the estimates V (A) and V (B)?

Predictor example

Episode 1	**A, 0, B, 0**
Episode 2	**B, 1**
Episode 3	**B, 1**
Episode 4	**B, 1**

Episode 5	**B, 0**
Episode 6	**B, 1**
Episode 7	**B, 1**
Episode 8	**B, 1**

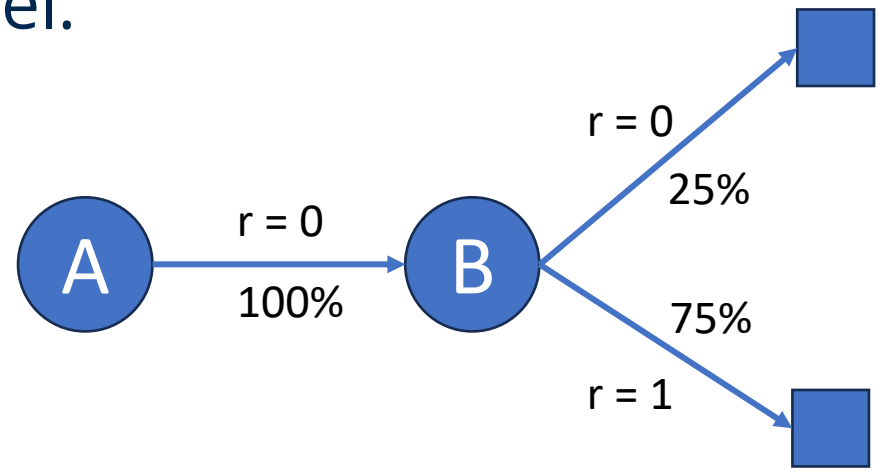
Model:

Predictor example

Episode 1	**A, 0, B, 0**
Episode 2	**B, 1**
Episode 3	**B, 1**
Episode 4	**B, 1**

Episode 5	**B, 0**
Episode 6	**B, 1**
Episode 7	**B, 1**
Episode 8	**B, 1**

Model:

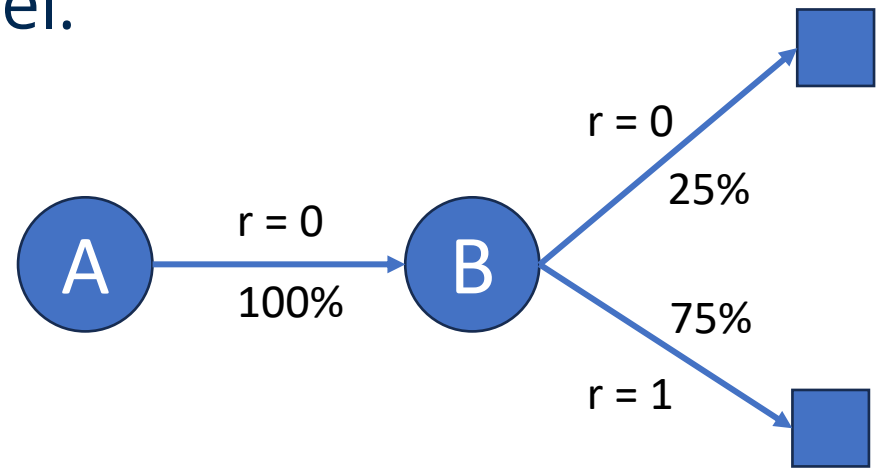


Predictor example

Episode 1	**A, 0, B, 0**
Episode 2	**B, 1**
Episode 3	**B, 1**
Episode 4	**B, 1**

Episode 5	**B, 0**
Episode 6	**B, 1**
Episode 7	**B, 1**
Episode 8	**B, 1**

Model:



Batch TD(0):

$V(A)$ =

$V(B)$ =

Batch MC:

$V(A)$ =

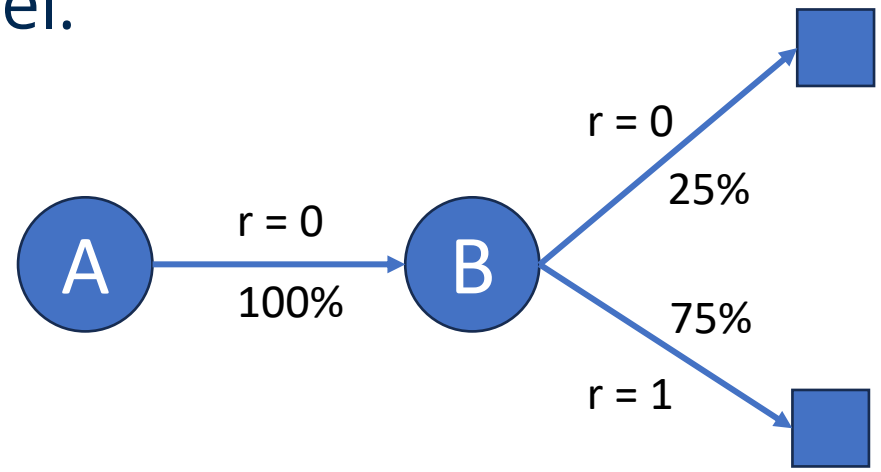
$V(B)$ =

Predictor example

Episode 1	**A, 0, B, 0**
Episode 2	**B, 1**
Episode 3	**B, 1**
Episode 4	**B, 1**

Episode 5	**B, 0**
Episode 6	**B, 1**
Episode 7	**B, 1**
Episode 8	**B, 1**

Model:



Batch TD(0):

$V(A) = ¾$

$V(B) = ¾$

First calculate $V(B)$ than calculate $V(A)$ based on it

Batch MC:

$V(A) = 0$

$V(B) = ¾$

Calculate each state based on how many times appeared and what was the result

Different estimates

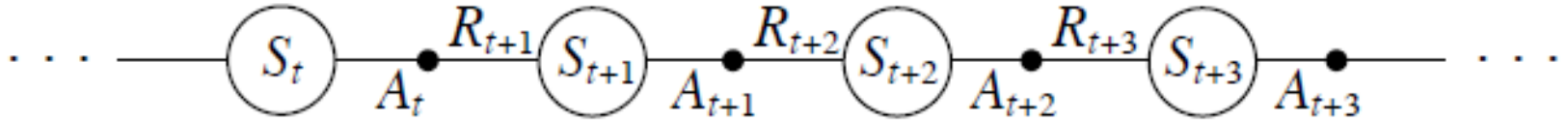
- Batch MC
  - finds the estimates that minimize mean-squared error
  - **maximum-likelihood estimate** of a parameter: is the parameter value whose probability of generating the data is greatest
- Batch TD
  - finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process
  - **certainty-equivalence estimate**: assuming that the estimate of the underlying process was known with certainty rather than being approximated

Different estimates

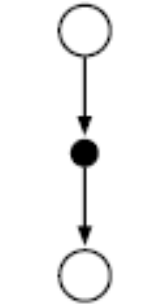
- Batch MC
  - finds the estimates that minimize mean-squared error
  - **maximum-likelihood estimate** of a parameter: is the parameter value whose probability of generating the data is greatest
- Batch TD
  - finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process
  - **certainty-equivalence estimate**: assuming that the estimate of the underlying process was known with certainty rather than being approximated

TD(0) is faster than MC methods because it computes the true certainty-equivalence estimate

Sarsa: On-policy TD Control

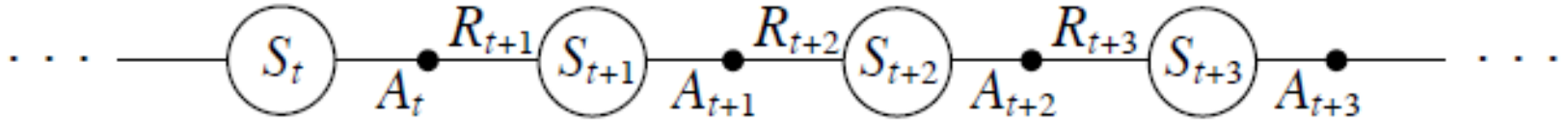


$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$



TD(0)

Sarsa: On-policy TD Control

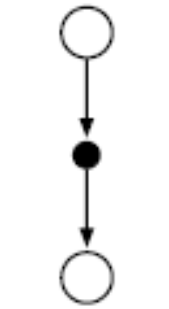


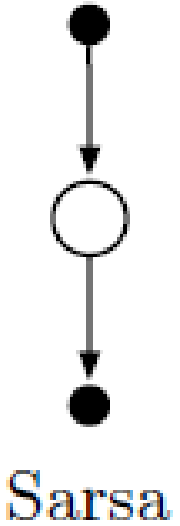
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

- Consider transitions from state-action pair to state-action pair, and learn the values of state-action pairs instead of state values

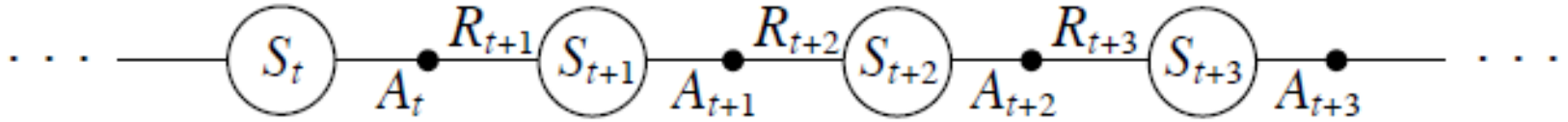
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

0 if S_{t+1} is terminal state

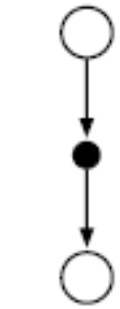




Sarsa: On-policy TD Control



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



TD(0)

- Consider transitions from state-action pair to state-action pair, and learn the values of state-action pairs instead of state values

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

0 if S_{t+1} is terminal state

Sarsa

Used elements: $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

SARSA

Pseudocode

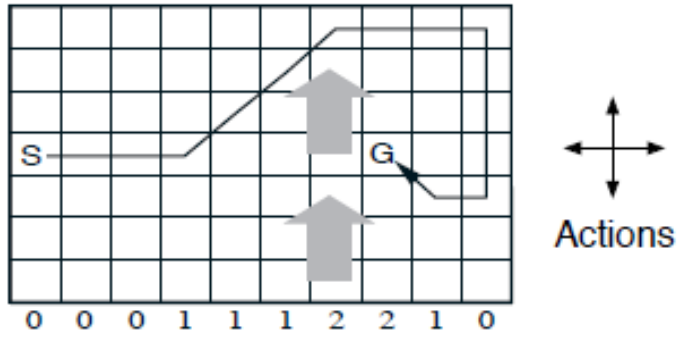
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

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

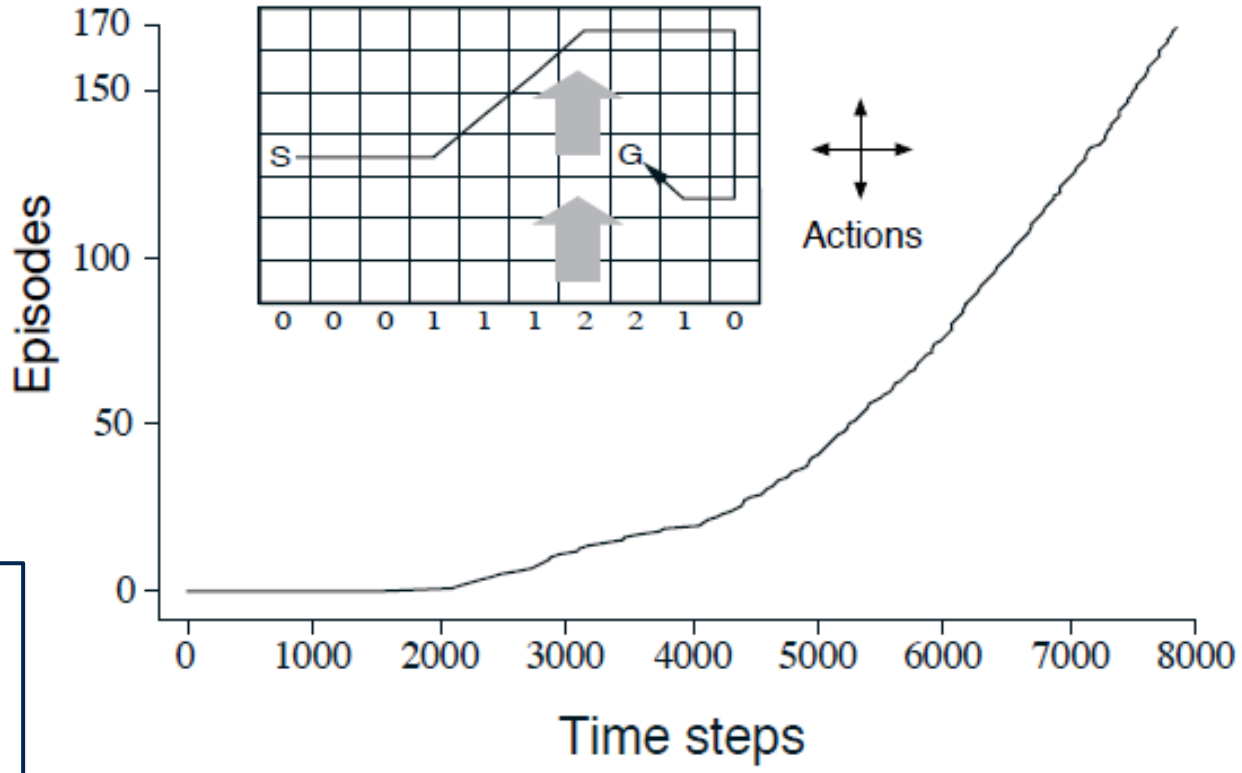
Windy gridworld

- undiscounted
- episodic task
- constant rewards of -1 until the goal state is reached
- 4 actions



Windy gridworld

- undiscounted
- episodic task
- constant rewards of -1 until the goal state is reached
- 4 actions



Note:
MC methods cannot easily be used on this task because termination is not guaranteed for all policies. If a policy was ever found that caused the agent to stay in the same state, then the next episode would never end

Sarsa do not have this problem

Q-learning: Off-policy TD Control

- Sarsa

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

Q-learning: Off-policy TD Control

- Sarsa

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

- Q-learning

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

  - The learned action-value function directly approximates q_*
  - Independent of the policy being followed
  - Minimal requirement of convergence:
    - all state-action pairs continue to be updated

Pseudocode

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

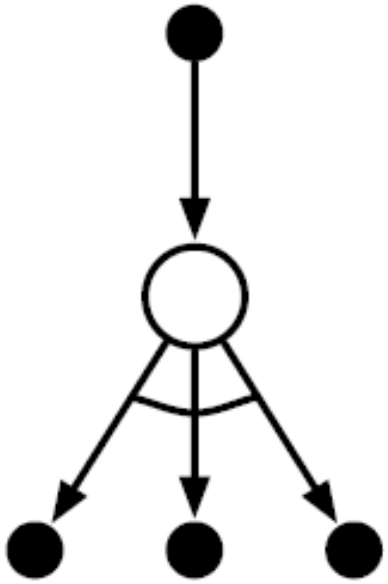
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

Backup diagram for Q-learning

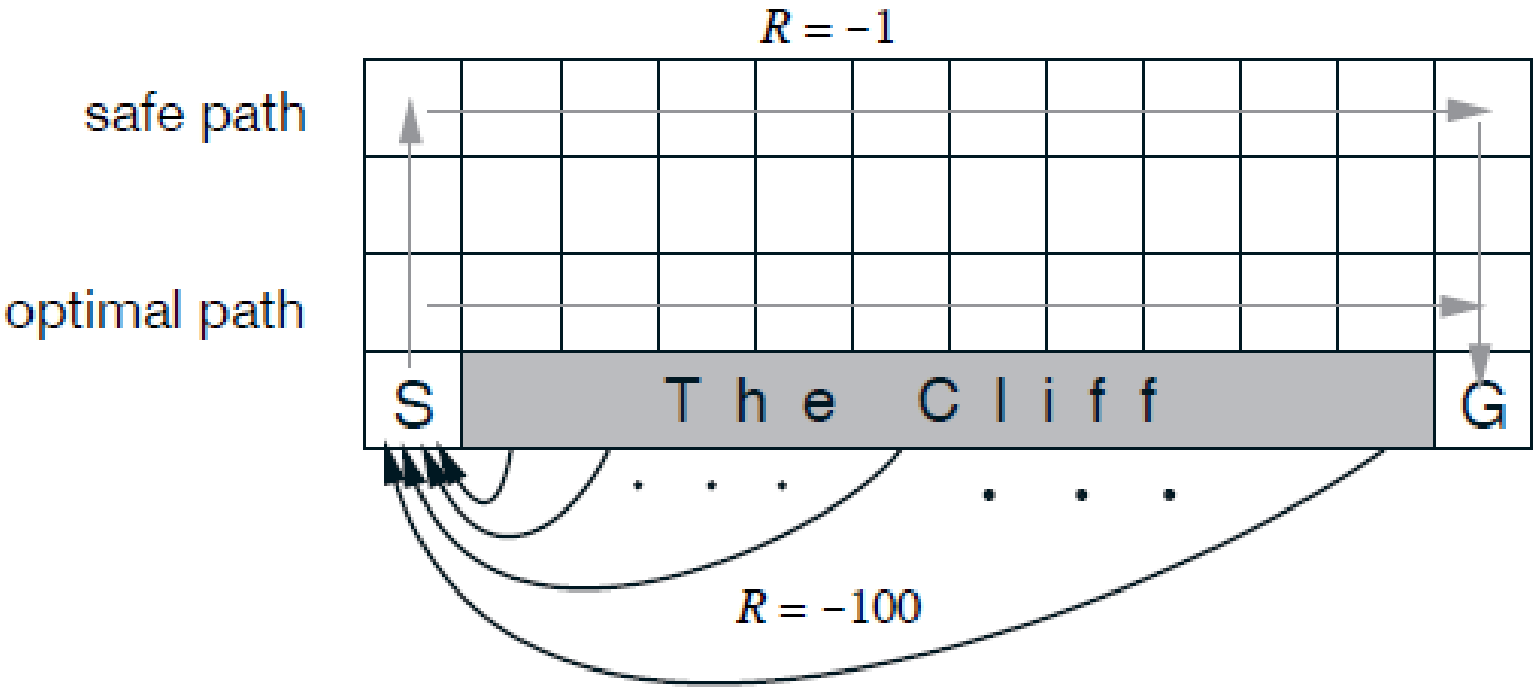
Backup diagram for Q-learning

- Updates a state-action pair, so the top node, must be a small, filled action node
- The update is also from action nodes
- Maximum of these " next action" nodes with an arc across



Q-learning

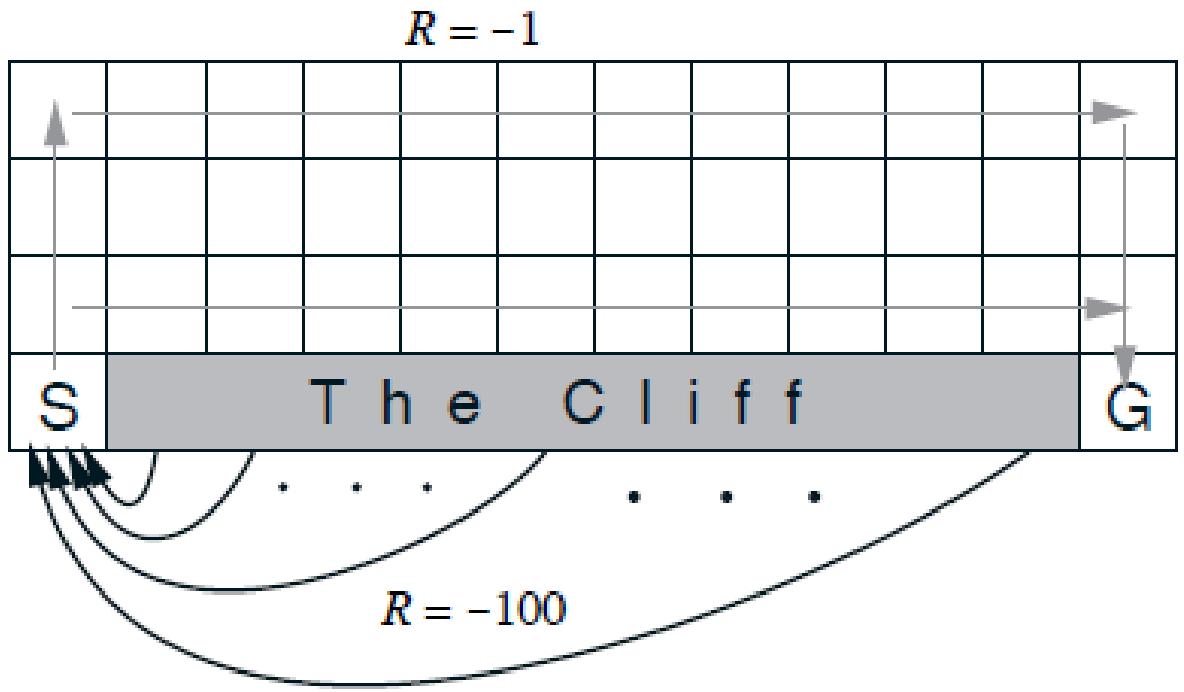
Cliff Walking example (Sarsa vs Q-learning)



Cliff Walking example (Sarsa vs Q-learning)

Sarsa

Q-learning



Q-learning: learns values for the optimal policy. This results in its occasionally falling off the cliff because of the ε-greedy action selection.
Sarsa: takes the action selection into account and learns the longer but safer path

Expected Sarsa

- Q-learning:

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

Expected Sarsa

- Q-learning:

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

- Expected Sarsa:

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

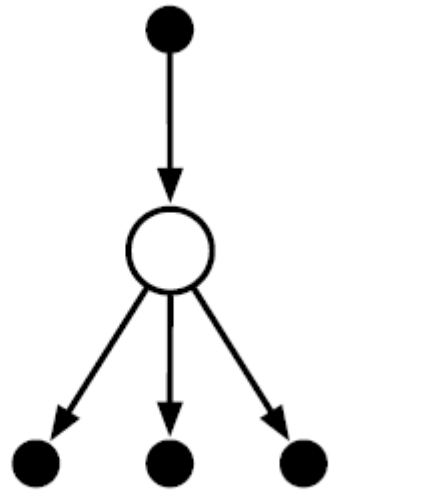
Instead of the maximum over next state-action pairs it uses the expected value, taking into account how likely each action is under the current policy

Backup diagram for Expected Sarsa

Backup diagram for Expected Sarsa

Comparison:

- **Expected Sarsa:**
  can safely set α=1 without suffering any degradation of asymptotic performance
- **Sarsa:**
  can only perform well in the long run at a small value of α, at which short-term performance is poor



Expected Sarsa

Maximization Bias

- the construction of target policies include maximization
- a maximum over estimated values is used implicitly as an estimate of the maximum value
- can lead to a significant positive bias

Maximization Bias

- the construction of target policies include maximization
- a maximum over estimated values is used implicitly as an estimate of the maximum value
- can lead to a significant positive bias

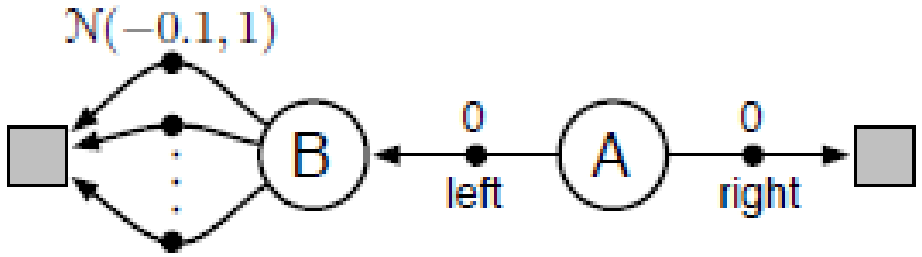
consider a single state s where there are many actions a whose true values, $q(s,a)$, are all zero but whose estimated values, $Q(s,a)$, are uncertain and thus distributed some above and some below zero. The maximum of the true values is zero, but the maximum of the estimates is positive, a positive bias

Maximization Bias

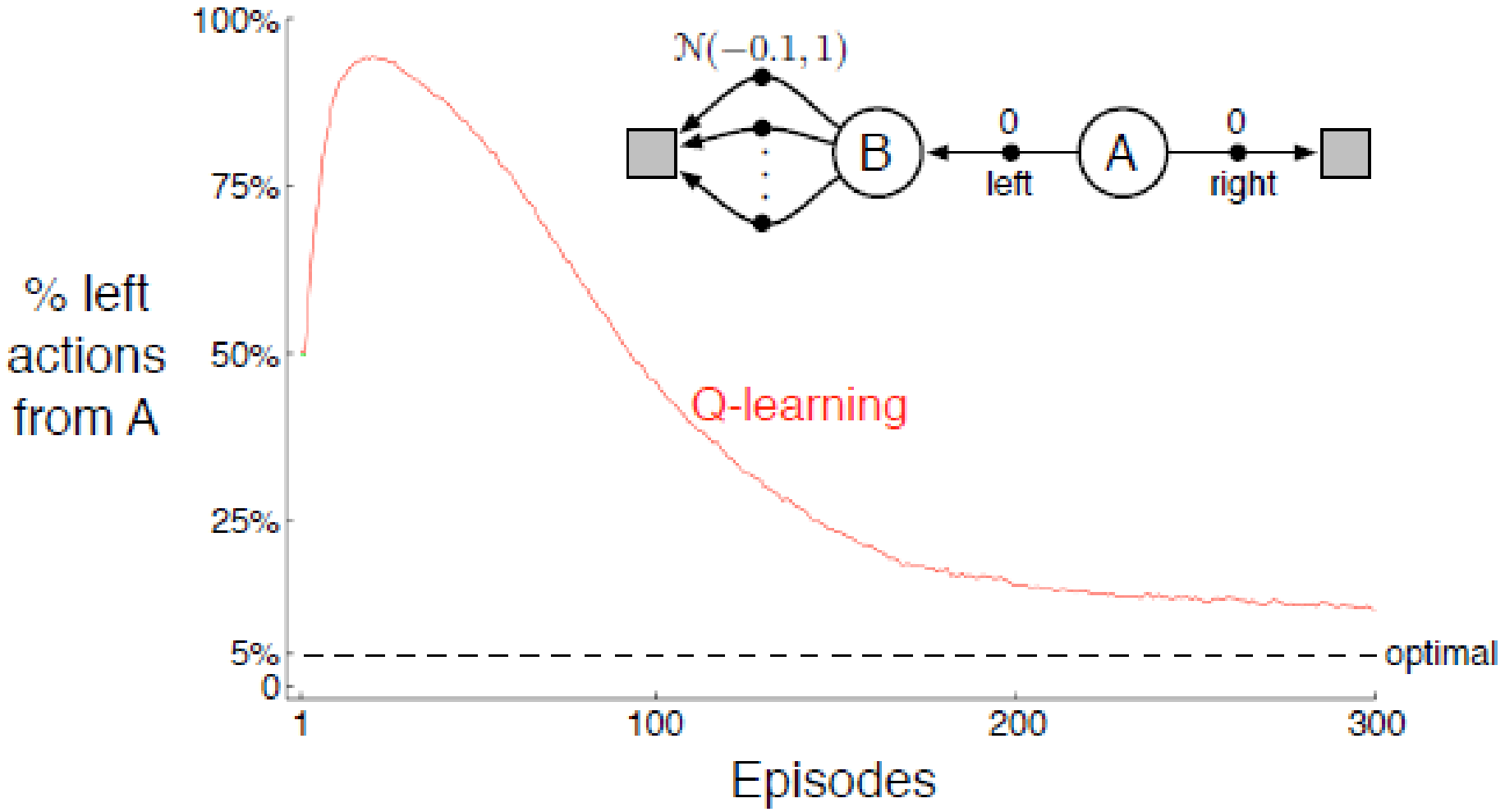
- the construction of target policies include maximization
- a maximum over estimated values is used implicitly as an estimate of the maximum value
- can lead to a significant positive bias

There will be a **positive maximization bias** if we use the maximum of the estimates as an estimate of the maximum of the true values

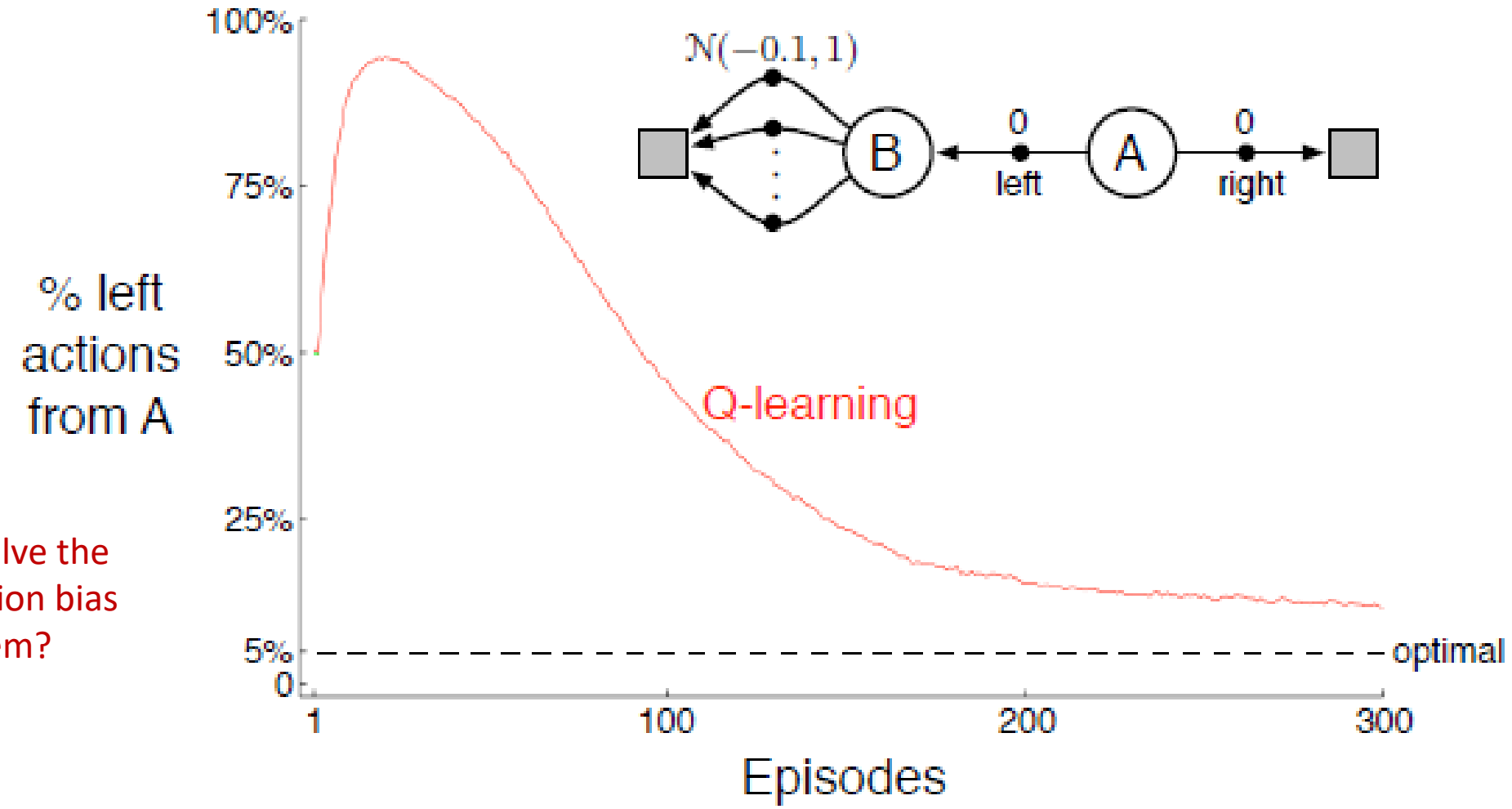
Maximization Bias Example



Maximization Bias Example

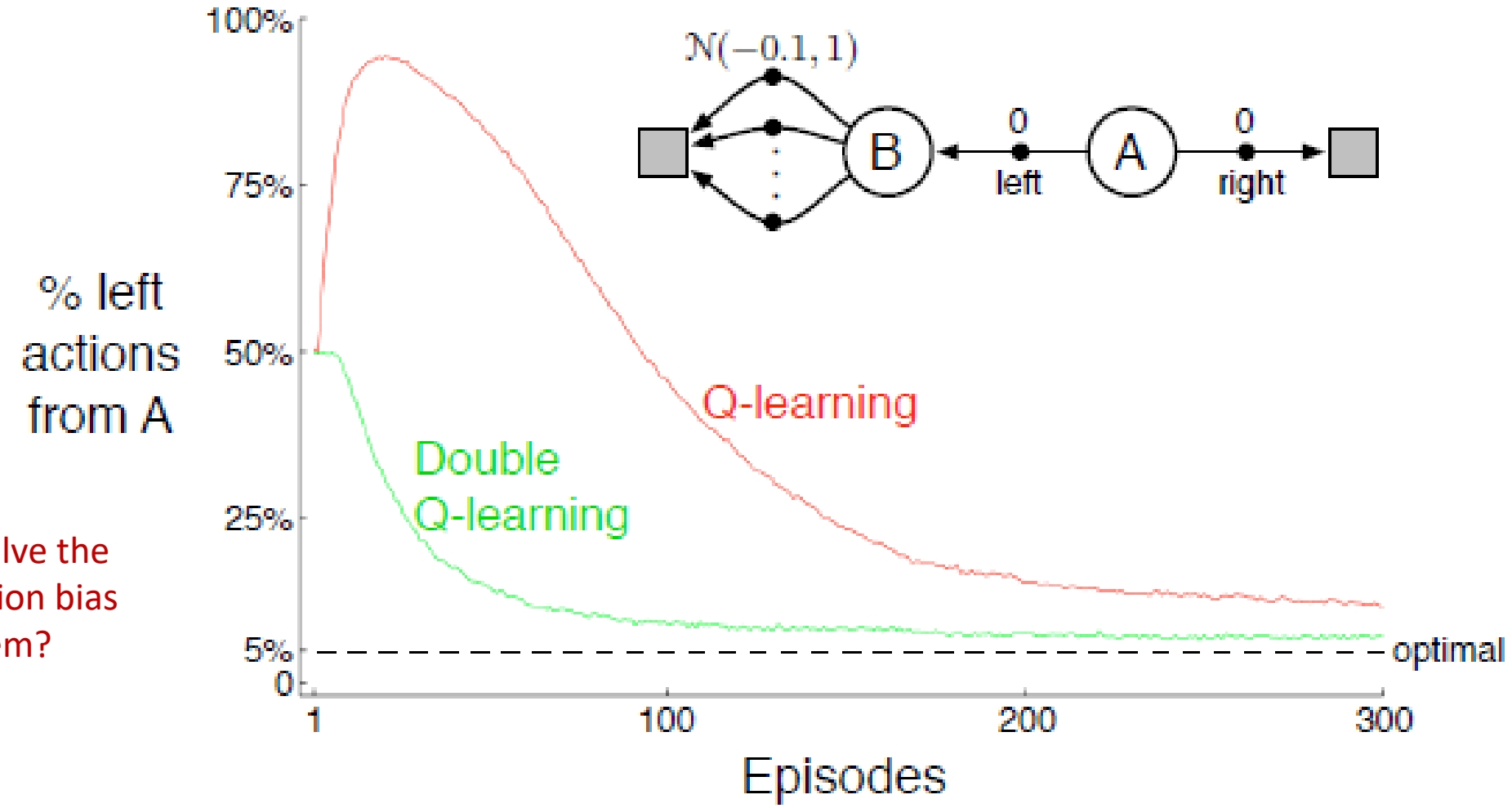


Maximization Bias Example



How to solve the maximization bias problem?

Maximization Bias Example



How to solve the maximization bias problem?

Double Learning

- Consider a bandit case with noisy estimates of the value of each of many actions obtained as sample averages of the rewards received on all the plays with each action

Double Learning

- Consider a bandit case with noisy estimates of the value of each of many actions obtained as sample averages of the rewards received on all the plays with each action
- **Problem**: positive maximization bias if we use the maximum of the estimates as an estimate of the maximum of the true values

Double Learning

- Consider a bandit case with noisy estimates of the value of each of many actions obtained as sample averages of the rewards received on all the plays with each action
- **Problem**: positive maximization bias if we use the maximum of the estimates as an estimate of the maximum of the true values
- **Old approach**: using the same samples (plays) both to determine the maximizing action and to estimate its value

Double Learning

- Consider a bandit case with noisy estimates of the value of each of many actions obtained as sample averages of the rewards received on all the plays with each action
- **Problem**: positive maximization bias if we use the maximum of the estimates as an estimate of the maximum of the true values
- **Old approach**: using the same samples (plays) both to determine the maximizing action and to estimate its value
- **New idea**: divided the plays in two sets and used them to learn two independent estimates

$$Q_1(a) \text{ and } Q_2(a)$$

Double Learning

- Use one estimate to determine the maximizing action

$$A^* = \text{argmax}_a \, Q_1(a)$$

- Use the other estimate to provide the estimate of its value

$$Q_2(A^*) = Q_2(\text{argmax}_a \, Q_1(a))$$

- This estimate will then be unbiased

$$\mathbb{E}[Q_2(A^*)] = q(A^*)$$

- Repeat the process with reversed roles

$$Q_1(\text{argmax}_a \, Q_2(a))$$

Double Learning

- Learn two estimates
- Only one estimate is updated on each play
- Doubles the memory requirements
- Does not increase the amount of computation per step
- Extends naturally to algorithms for full MDPs

Double Learning

- Learn two estimates
- Only one estimate is updated on each play
- Doubles the memory requirements
- Does not increase the amount of computation per step
- Extends naturally to algorithms for full MDPs

The double learning algorithm analogous to Q-learning, called **Double Q-learning**, divides the time steps in two, perhaps by flipping a coin on each step.
If the coin comes up heads, the update is:

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

If the coin comes up tails, then the same update is done with Q1 and Q2 switched.

Pseudocode

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy 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:

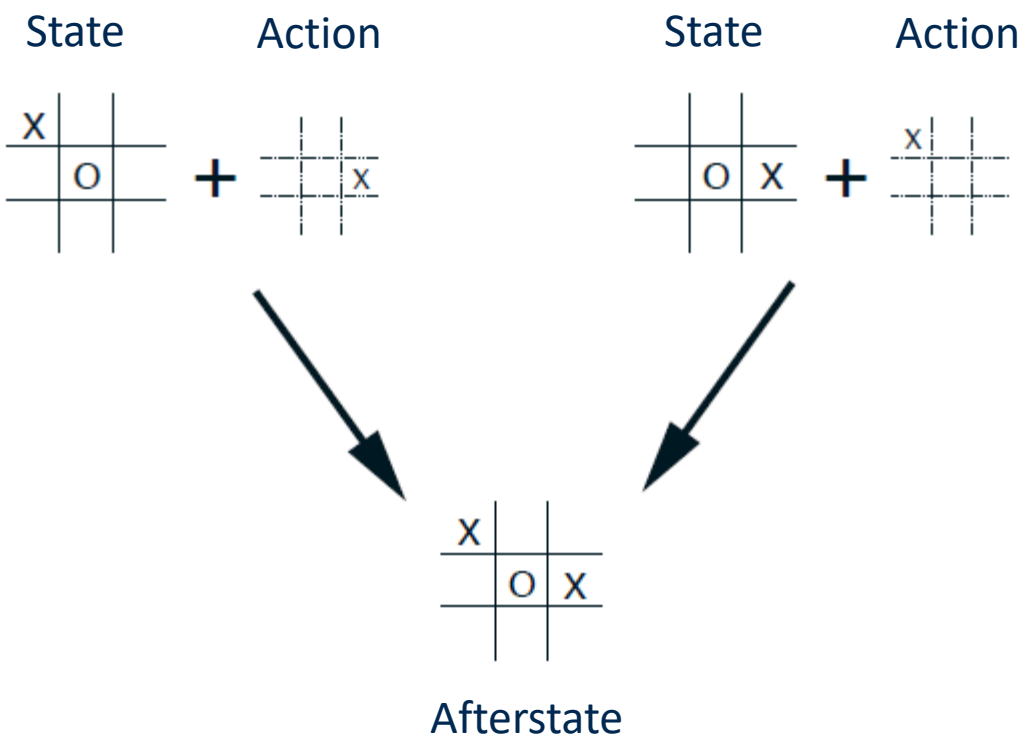
$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

The behavior policy can use both action-value estimates

Afterstate

- conventional **state-value function** evaluates states in which the agent has the option of selecting an action
- **afterstate-value function** evaluates board positions after the agent has made its move



Summary

- So far
  - One-step, tabular, model-free TD methods
- What is next?
  - Extend to *n*-step form
  - Use function approximation instead of tables
    - Include neural networks and deep learning

ELTE | FACULTY OF INFORMATICS

Thank you for your attention!