# REINFORCEMENT LEARNING

Seminar @ UPC TelecomBCN Barcelona (2nd edition). Spring 2020.

## Instructors

Josep Vidal

Margarita Cabrera

Xavier Giró-i-Nieto

## Organizers

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

## Supporters

Google Cloud

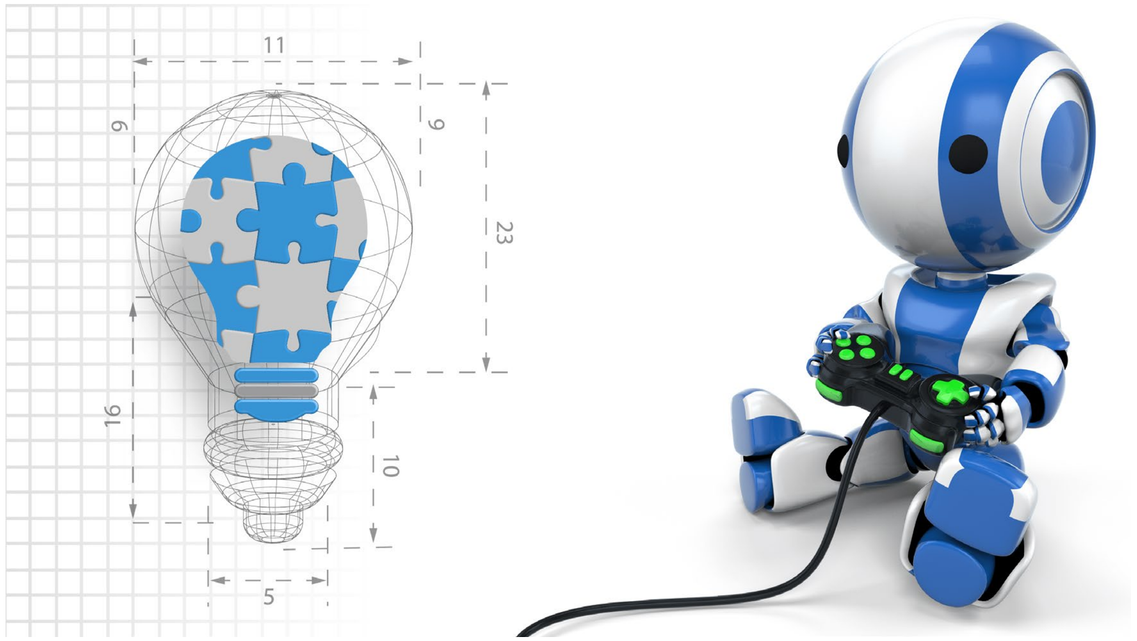
GitHub Education

+ info: http://bit.ly/upcrl-2020

telecos BCN

UPC

## 5. Temporal-difference learning



## Motivation

MC does not work for scenarios without termination

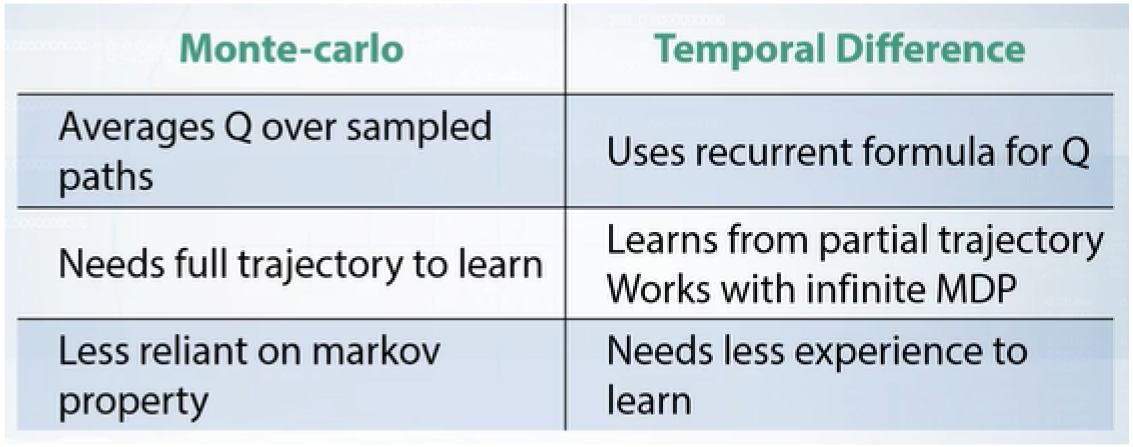
It updates only at the end of the episode (sometimes - it is too late)

It cannot use bootstrapping

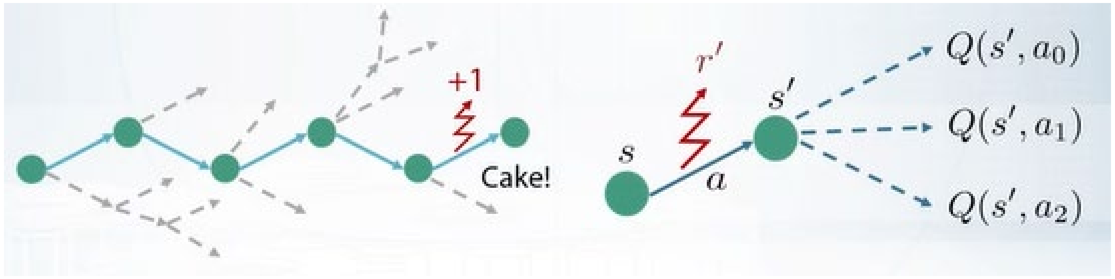
Temporal Difference Learning

- Learning after time steps, not on entire episodes: central idea in RL.
- Combination of Monte Carlo and dynamic programming:
  - Do not need a model, learn from experience.
  - Update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).

# Motivation

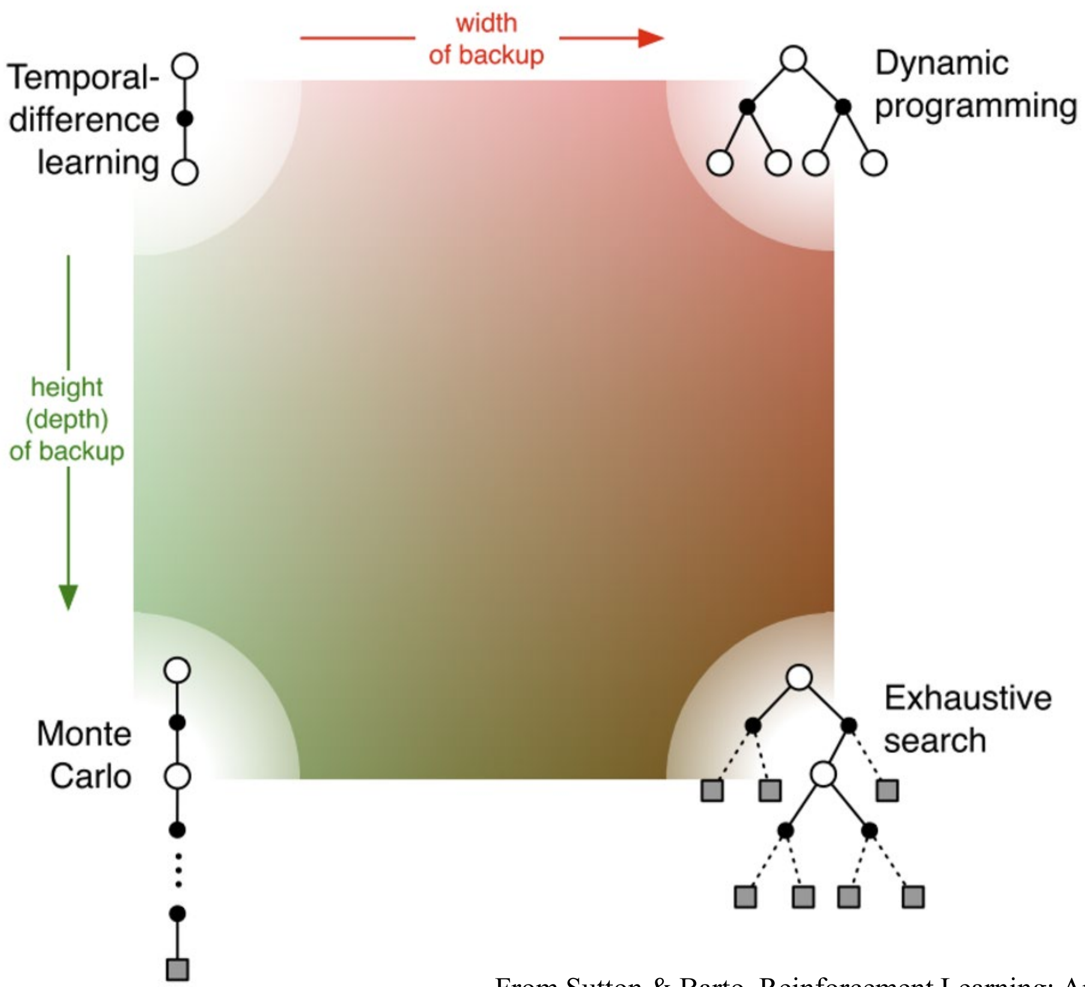


| Monte-carlo | Temporal Difference |
|---|---|
| Averages Q over sampled paths | Uses recurrent formula for Q |
| Needs full trajectory to learn | Learns from partial trajectory<br>Works with infinite MDP |
| Less reliant on markov property | Needs less experience to learn |



Closer to how humans learn

## TD learning and other methods



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

## TD Prediction

MC: uses sampling

$$\begin{aligned} v_\pi(s) &\doteq E_\pi\left\{G_t \middle| S_t = s\right\} \\ &= E_\pi\left\{R_{t+1} + \gamma G_{t+1} \middle| S_t = s\right\} \\ &= E_\pi\left\{R_{t+1} + \gamma v_\pi(S_{t+1}) \middle| S_t = s\right\} \end{aligned}$$

DP: uses the current estimate of $V(S_{t+1})$ instead of true value $v_\pi(S_{t+1})$

TD: uses both ideas

- MC updating error: $\delta_t^{MC} \doteq G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_t^{TD}$

- TD updating error: $\delta_t^{TD} \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

## TD Prediction

Let us use the running mean to replace expectation with sampling:

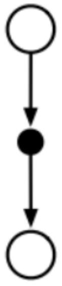
- Every-visit constant-$\alpha$ MC method: $V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right]$
- One-step TD: $V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$

This is a non-biased noisy version of expectation.

α is the learning rate: if it is too small, the convergence time is longer.

If $\gamma$ is close to 1, actions adopted at time *t* are highly probable to result in further favourable states in time *t*+1.

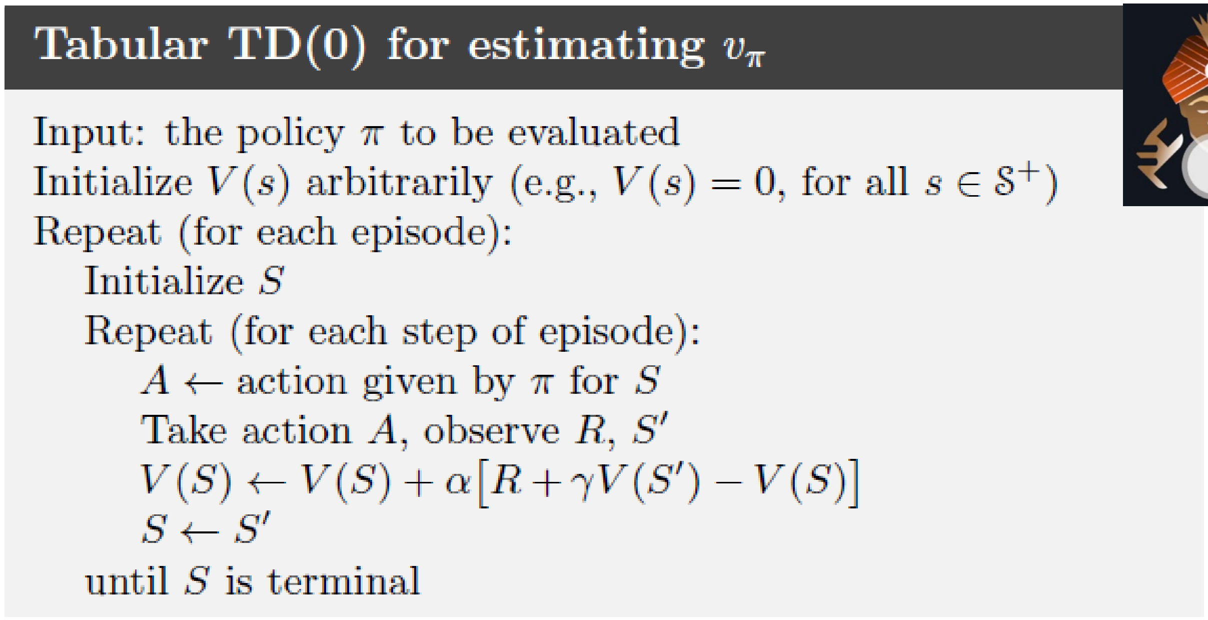
The backup diagram…



## TD Prediction

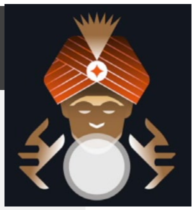
Let us use the running mean to replace expectation with sampling:

- Every-visit constant-$\alpha$ MC method: $V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right]$
- One-step TD: $V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$



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

```
Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
```



## Convergence of TD

- TD converges with probability 1 to $v_\pi$ if the step size satisfies the conditions:

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

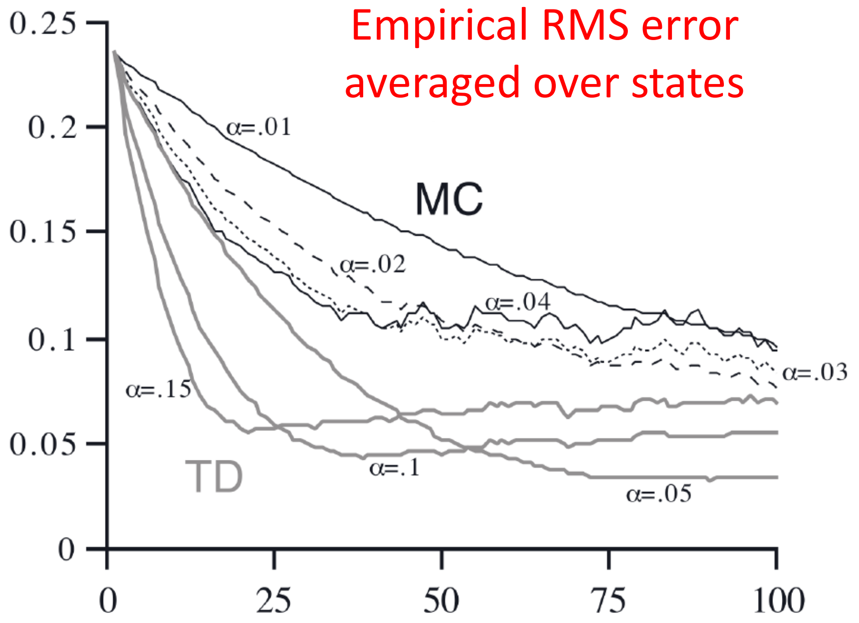
where $\alpha_n(a)$ is the step size for action $a$ at step $n$.

## Advantages of TD Prediction

- TD is implemented in an on-line, fully incremental fashion.
- Which method learns faster (TD or MC)?

  - Open question: no one has been able to prove mathematically that one method converges faster than the other.

  - In practice, TD methods have usually been found to converge faster than constant- MC methods on stochastic tasks.



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

## Control using TD: SARSA

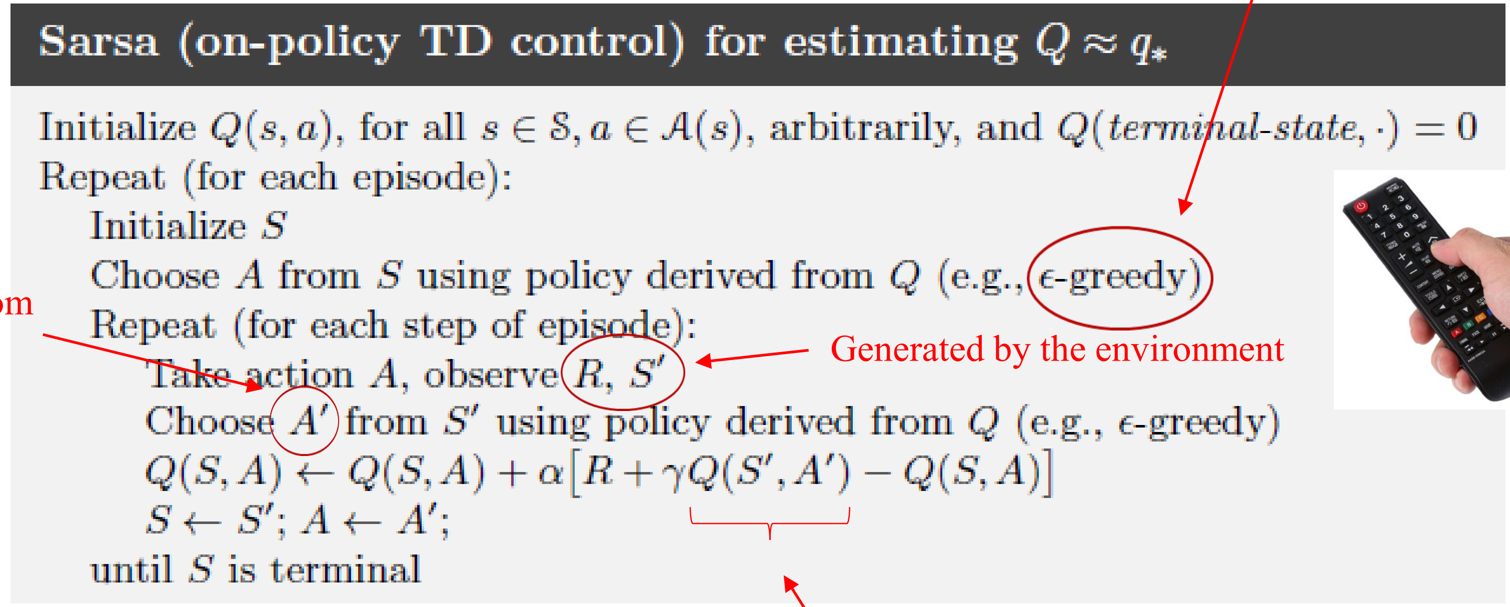
It is an on-policy method

- Learn $q_\pi(s, a)$ rather than $v_\pi(s)$ function.
- For the current behavior policy $\pi$ and for all states $s$ and actions $a$…

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

- This update is done after every transition from a non-terminal state $S_t$.
- This rule uses every element of the quintuple of events: $S_t\, A_t\, R_{t+1}\, S_{t+1}\, A_{t+1}$

Randomize actions



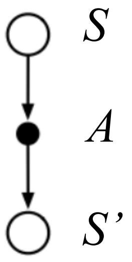
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

Random

Generated by the environment

Compare to Q-learning
(next slide)



How do we select the optimum policy?

## Control using TD: Q-learning

It is an off-policy method

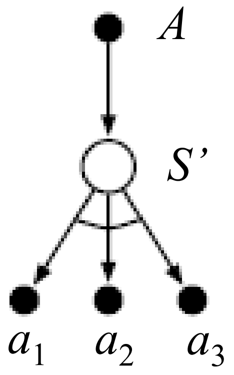
The learnt action-value function $Q$ directly approximates $q^*$, the optimal action-value function, from a policy that is not the one taken by agent.

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

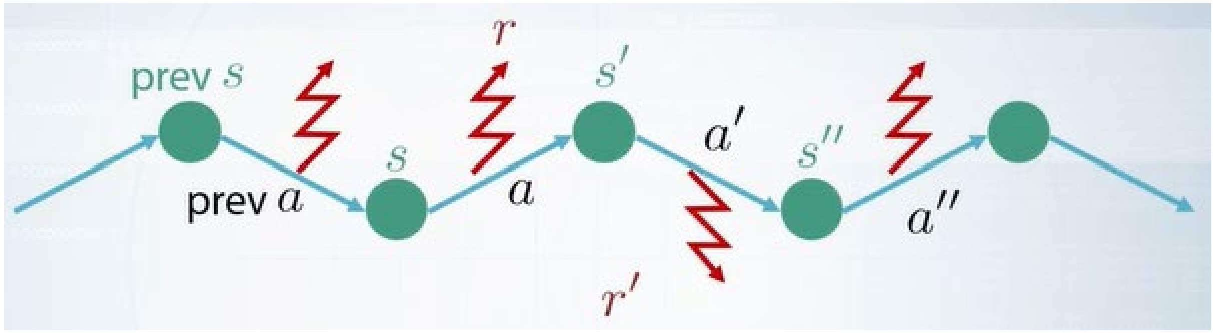
**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

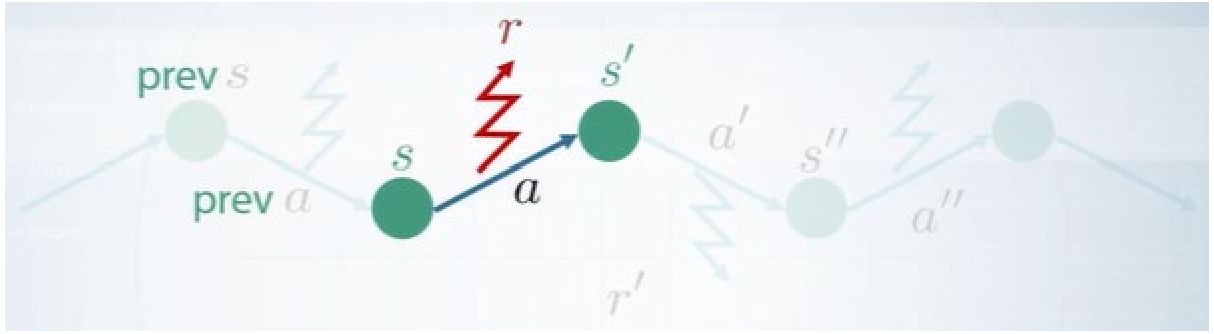
```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $\boxed{R, S'}$ ← Generated by the environment
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \underbrace{\max_a Q(S',a)}_{} - Q(S,A)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

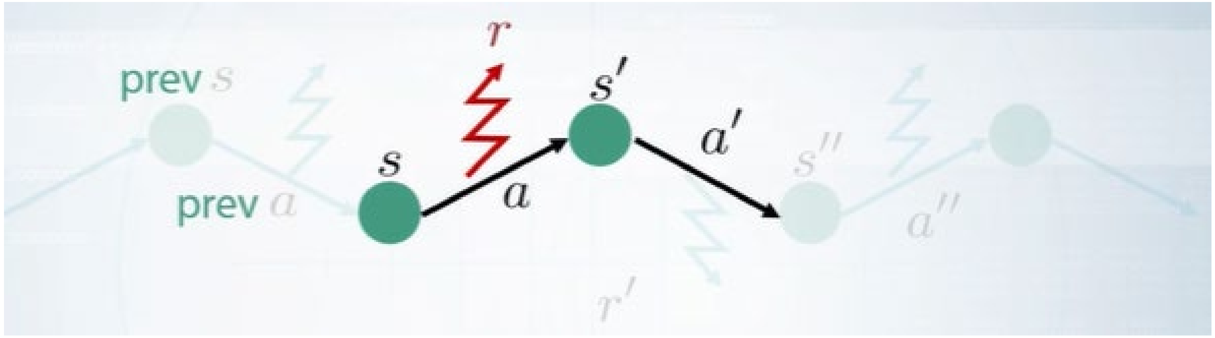
Q assuming the agent takes the optimum policy, although it does not (compare to SARSA)



## MC vs one-step TD vs SARSA

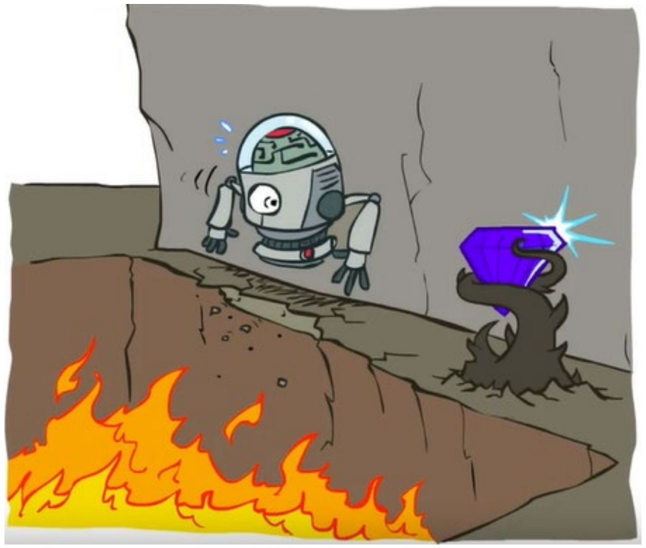


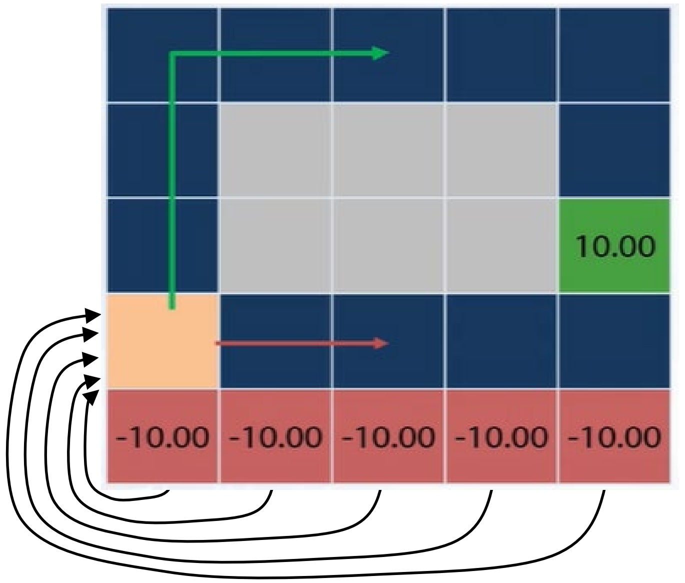




## Example 5.1: Gridworld over a cliff

An agent in a gridworld can choose between two paths before arriving to a big reward position: the red path is shorter but close to a cliff. The green one is safer but longer.





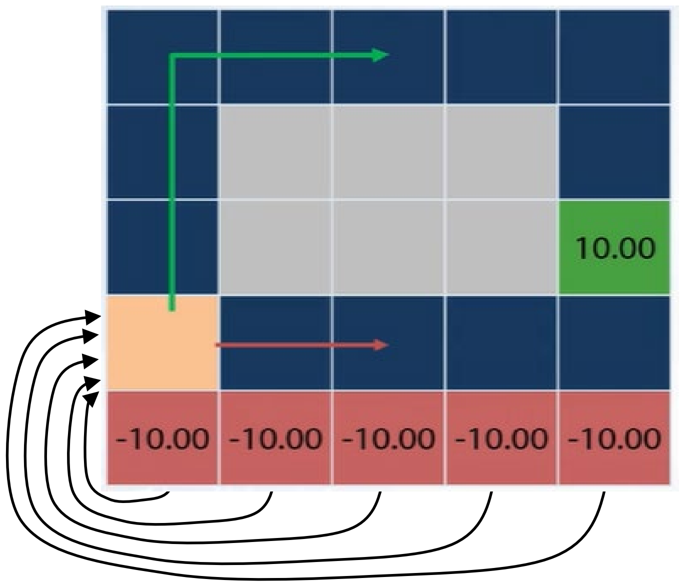
Assume we are using an exploratory Q-learning method where $\gamma = 0.99$, $\varepsilon = 0.1$, and there is no slipping effect when walking in the red path.

Which path will Q-learning learn? The red one, maximizes the reward.

From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

## Example 5.1: Gridworld over a cliff

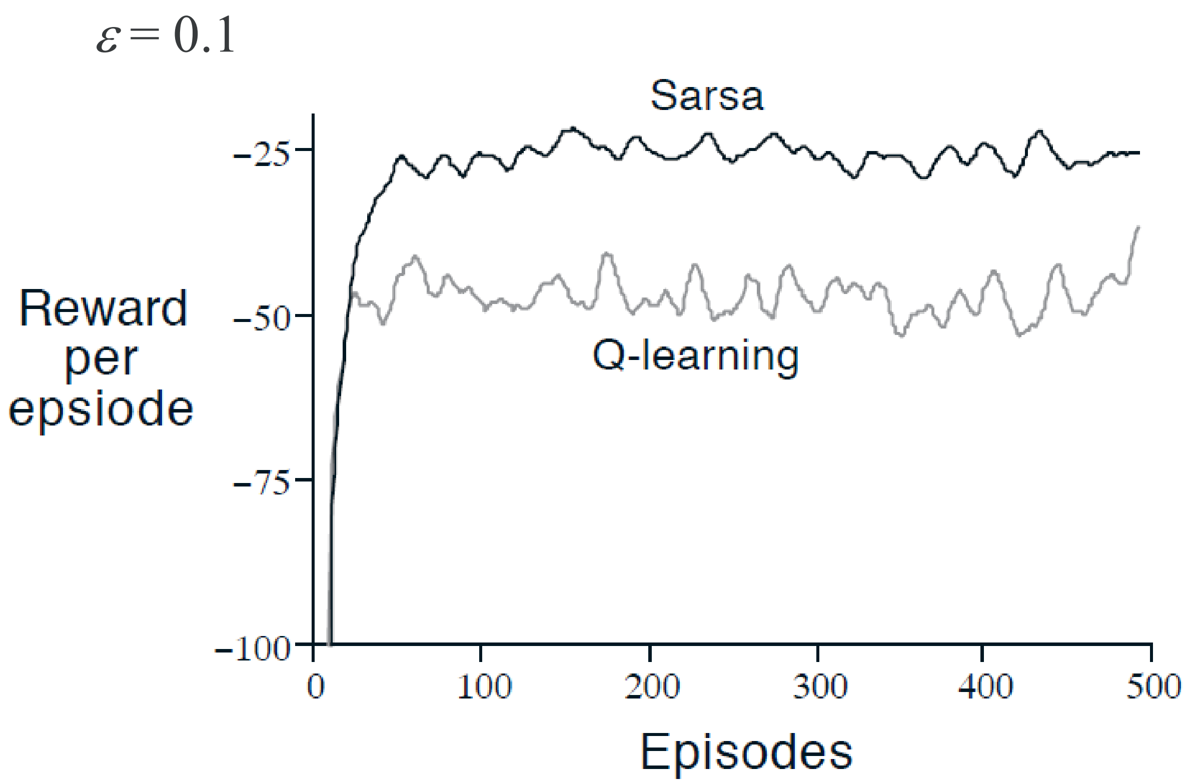
However, due to $\varepsilon$-greedy action taking the agent falls off the cliff occasionally. But we cannot get rid of exploration because the environment might be changing.



How can we keep exploring and avoiding taking destructive actions?

This is avoided using SARSA, where $Q(s,a)$ is used instead of $\max_a(Q(s,a))$. It takes the action selection into account. SARSA will learn the green path.

## Example 5.1: Gridworld over a cliff



Q-learning has worse on-line performance, although both would learn the optimum path if $\varepsilon$ is taken to zero.

From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

## Matlab Reinforcement Learning Tool

- RL Tool

https://es.mathworks.com/help/reinforcement-learning/index.html?s_tid=CRUX_lftnav

- RL Agents

https://es.mathworks.com/help/reinforcement-learning/ug/create-agents-for-reinforcement-learning.html

- Q-Learning Agents

https://es.mathworks.com/help/reinforcement-learning/ug/q-agents.html#responsive_offcanvas

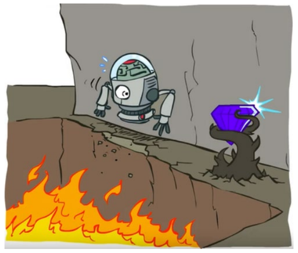
Training Algorithm: it is the Q-learning as in slide 12 of chapter 5

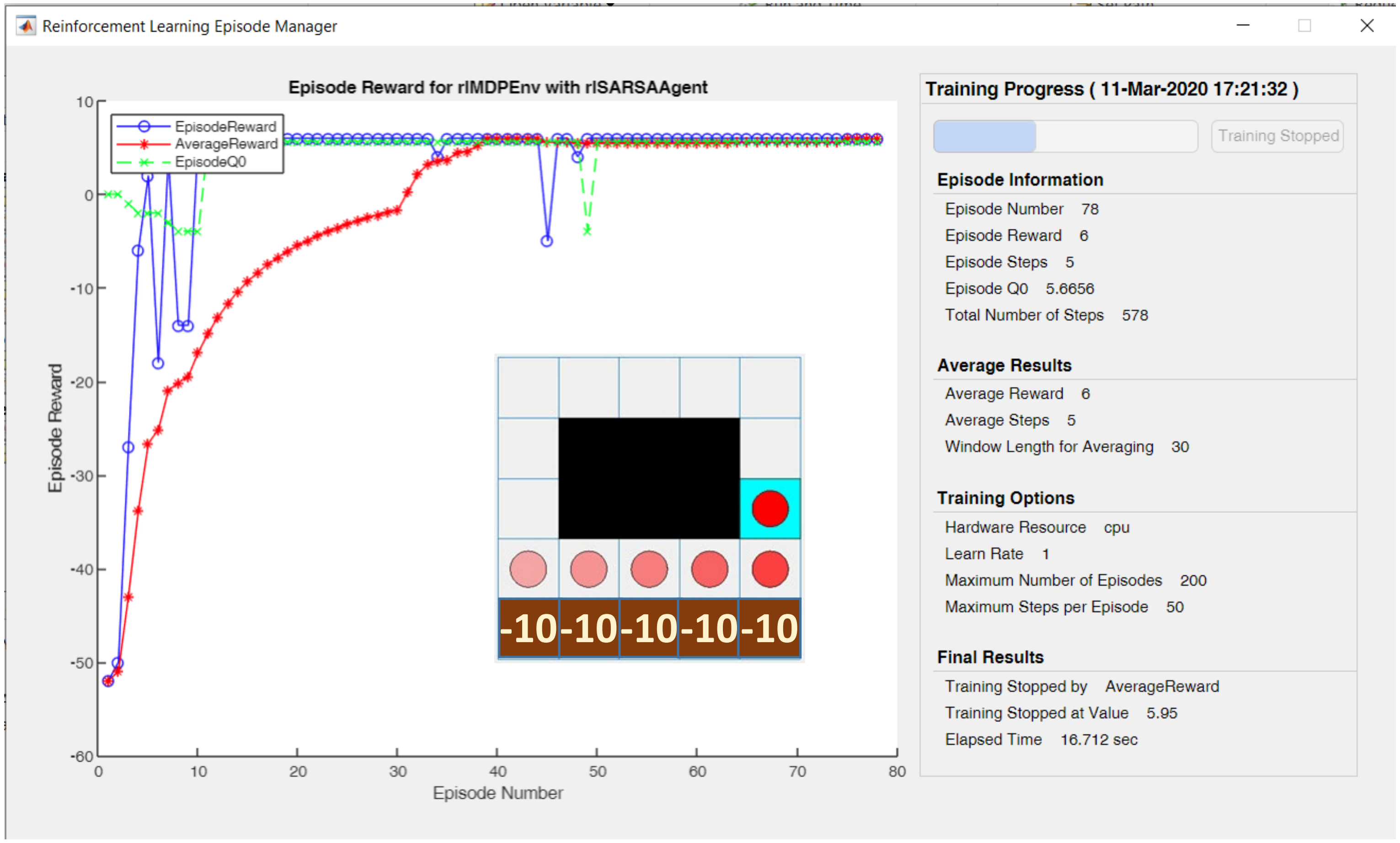
- SARSA Agents

https://es.mathworks.com/help/reinforcement-learning/ug/sarsa-agents.html

- Training Algorithm: it is the SARSA as in slide 11 of chapter 5

## Training tool **SARSA.** Cliff example reward of standard cells *r* = **-1**





**Training Progress ( 11-Mar-2020 17:21:32 )**

**Episode Information**

| | |
|---|---|
| Episode Number | 78 |
| Episode Reward | 6 |
| Episode Steps | 5 |
| Episode Q0 | 5.6656 |
| Total Number of Steps | 578 |

**Average Results**

| | |
|---|---|
| Average Reward | 6 |
| Average Steps | 5 |
| Window Length for Averaging | 30 |

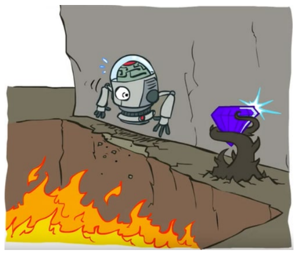
**Training Options**

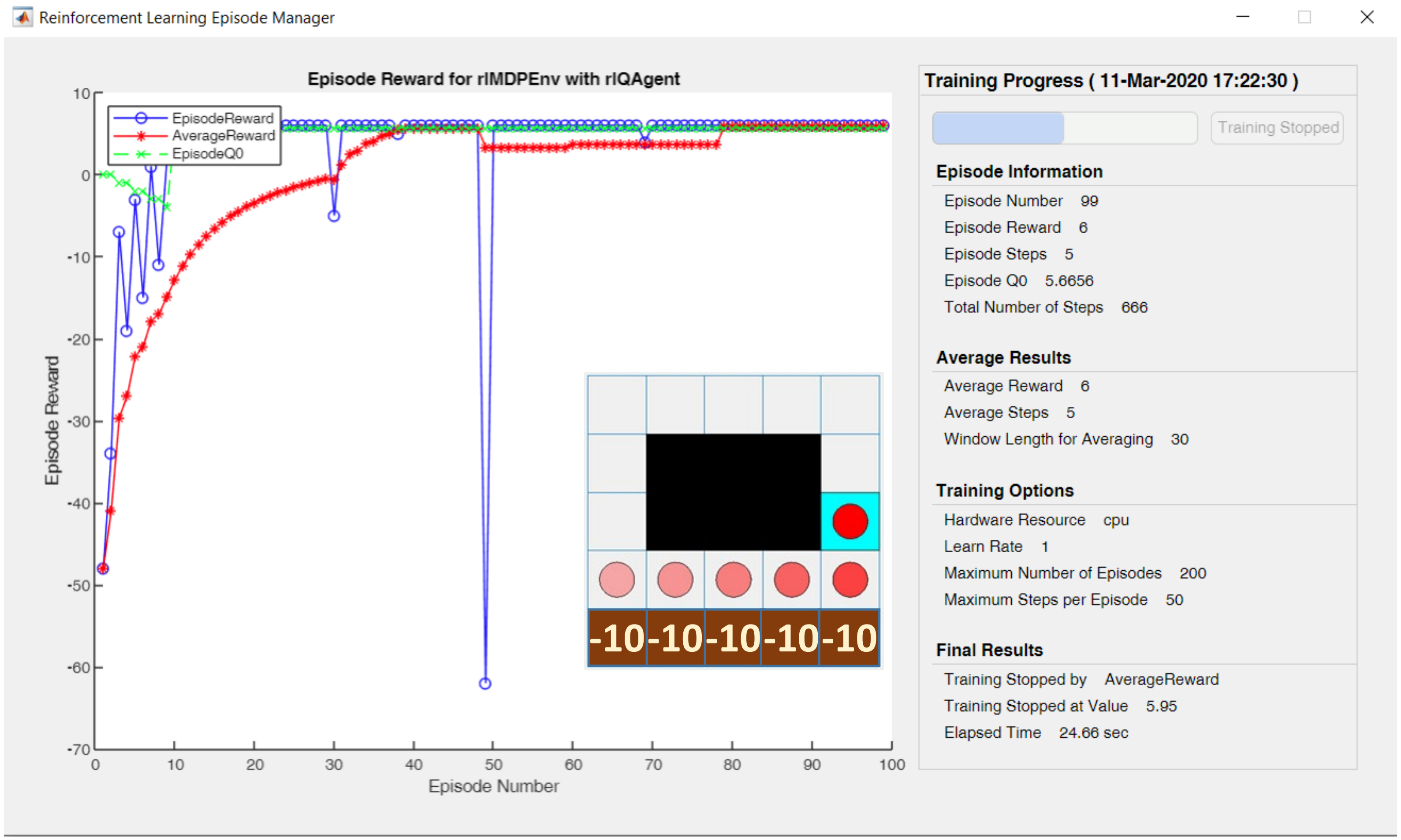
| | |
|---|---|
| Hardware Resource | cpu |
| Learn Rate | 1 |
| Maximum Number of Episodes | 200 |
| Maximum Steps per Episode | 50 |

**Final Results**

| | |
|---|---|
| Training Stopped by | AverageReward |
| Training Stopped at Value | 5.95 |
| Elapsed Time | 16.712 sec |

## Training tool **Q-Learning.** Cliff example, ***r* = -1**




Reinforcement Learning Episode Manager
Episode Reward for rlMDPEnv with rlQAgent
EpisodeReward
AverageReward
EpisodeQ0
Episode Reward
Episode Number
-10
-10
-10
-10
-10

Training Progress ( 11-Mar-2020 17:22:30 )

Training Stopped

**Episode Information**

| | |
|---|---|
| Episode Number | 99 |
| Episode Reward | 6 |
| Episode Steps | 5 |
| Episode Q0 | 5.6656 |
| Total Number of Steps | 666 |

**Average Results**

| | |
|---|---|
| Average Reward | 6 |
| Average Steps | 5 |
| Window Length for Averaging | 30 |

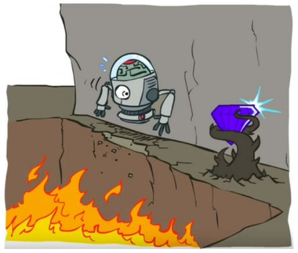
**Training Options**

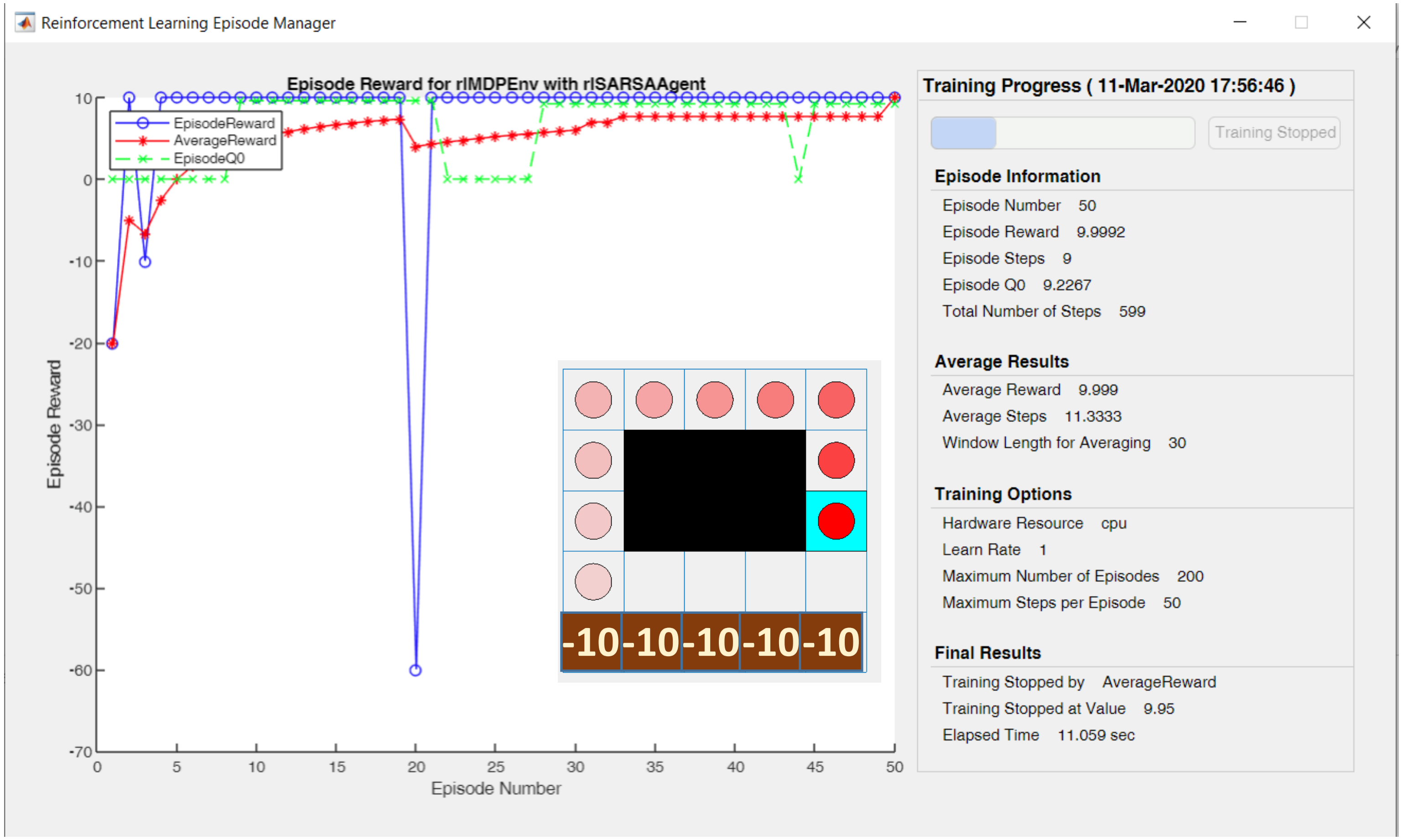
| | |
|---|---|
| Hardware Resource | cpu |
| Learn Rate | 1 |
| Maximum Number of Episodes | 200 |
| Maximum Steps per Episode | 50 |

**Final Results**

| | |
|---|---|
| Training Stopped by | AverageReward |
| Training Stopped at Value | 5.95 |
| Elapsed Time | 24.66 sec |

## Training tool **SARSA.** Cliff example, ***r* = -0.0001**





**Training Progress ( 11-Mar-2020 17:56:46 )**

**Episode Information**

Episode Number 50
Episode Reward 9.9992
Episode Steps 9
Episode Q0 9.2267
Total Number of Steps 599

**Average Results**

Average Reward 9.999
Average Steps 11.3333
Window Length for Averaging 30

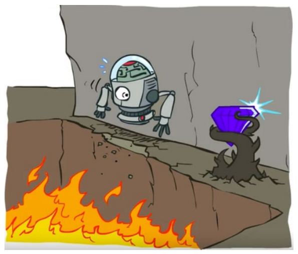
**Training Options**

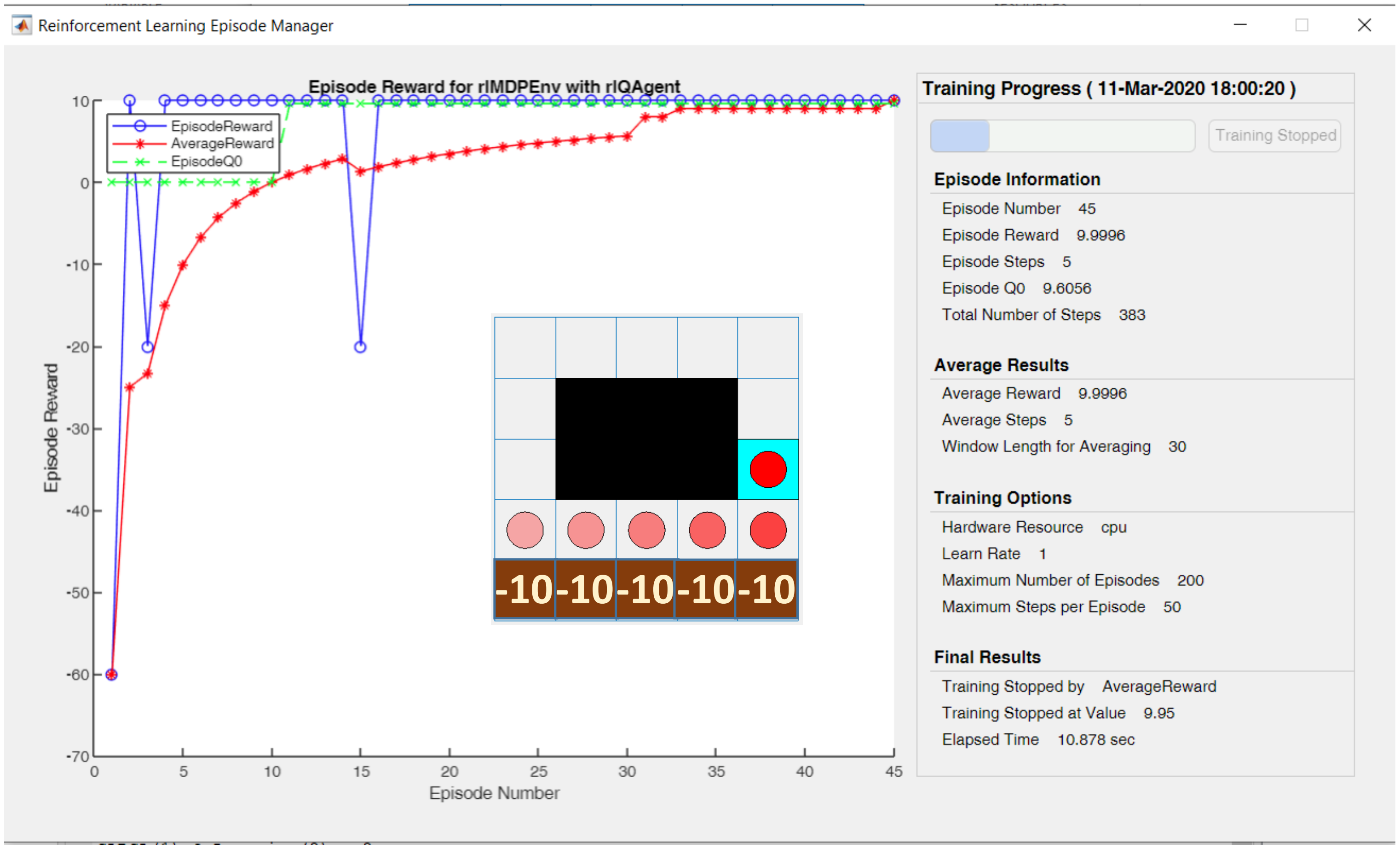
Hardware Resource cpu
Learn Rate 1
Maximum Number of Episodes 200
Maximum Steps per Episode 50

**Final Results**

Training Stopped by AverageReward
Training Stopped at Value 9.95
Elapsed Time 11.059 sec

## Training tool **Q-Learning.** Cliff example, ***r* = -0.0001**





**Training Progress ( 11-Mar-2020 18:00:20 )**

**Episode Information**

| | |
|---|---|
| Episode Number | 45 |
| Episode Reward | 9.9996 |
| Episode Steps | 5 |
| Episode Q0 | 9.6056 |
| Total Number of Steps | 383 |

**Average Results**

| | |
|---|---|
| Average Reward | 9.9996 |
| Average Steps | 5 |
| Window Length for Averaging | 30 |

**Training Options**

| | |
|---|---|
| Hardware Resource | cpu |
| Learn Rate | 1 |
| Maximum Number of Episodes | 200 |
| Maximum Steps per Episode | 50 |

**Final Results**

| | |
|---|---|
| Training Stopped by | AverageReward |
| Training Stopped at Value | 9.95 |
| Elapsed Time | 10.878 sec |

# Expected SARSA

- Use the expected value instead of the maximum over next state/action pairs, taking into account how likely each action is under the current policy $\pi$ (off-policy method if $\pi$ is different from the behavioral policy):

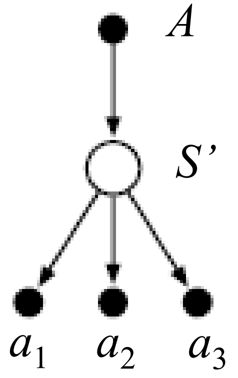
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma E_\pi\{Q(S_{t+1},A_{t+1}) | S_{t+1}\} - Q(S_t,A_t)\right]$$

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

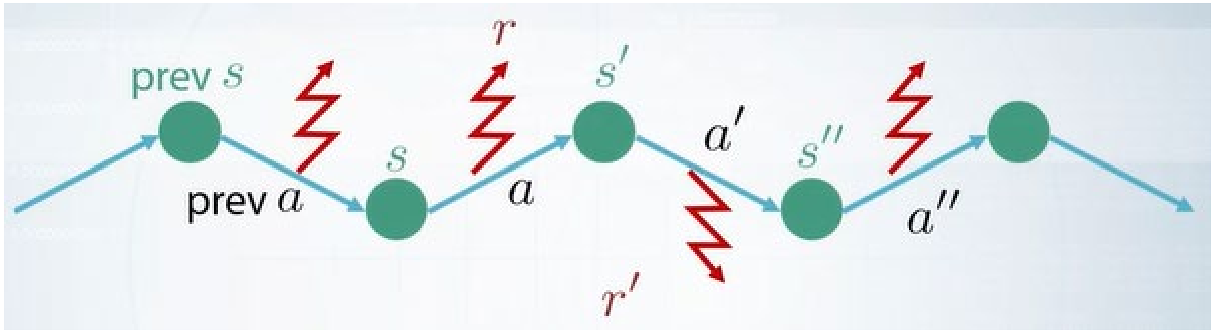
Compare to previous methods

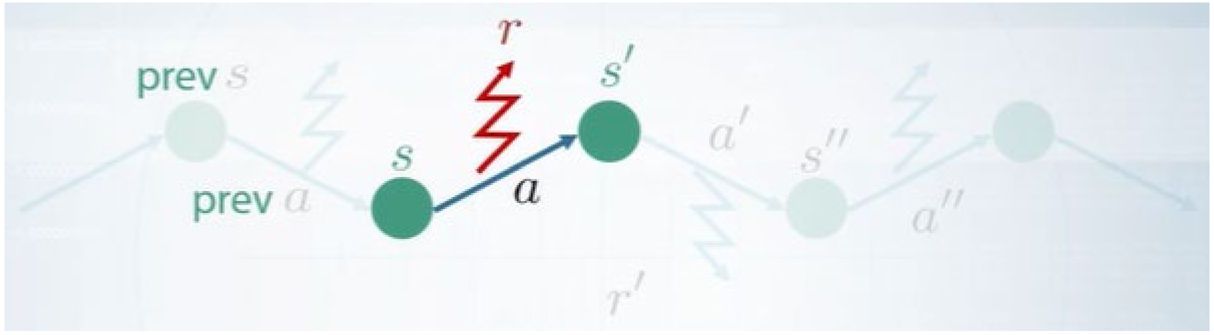
- Given $S_{t+1}$ this algorithm moves deterministically in the same direction as SARSA moves in expectation.

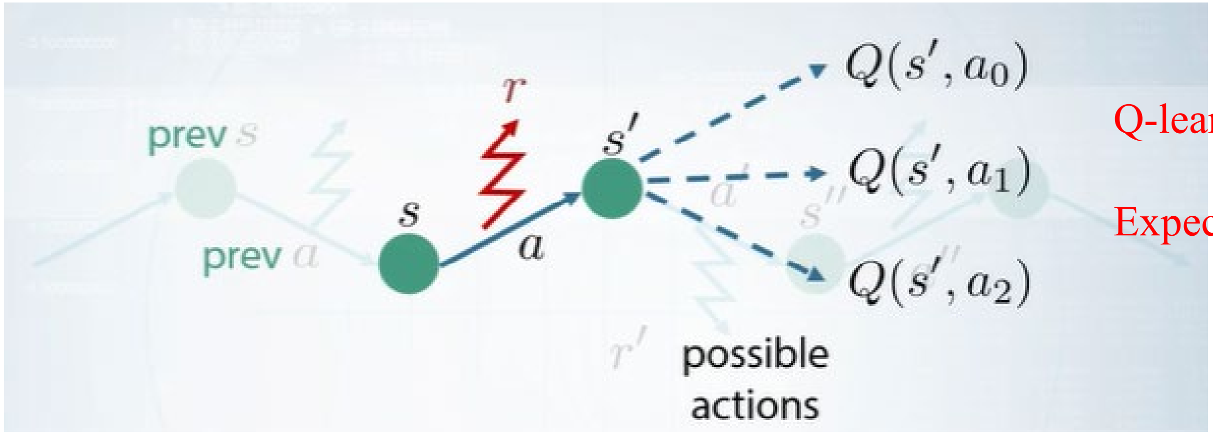
- Outperforms other TD methods: it eliminates the variance associated to the random selection of $A'$ on a Q function possibly poorly estimated.



## MC vs one-step TD vs …



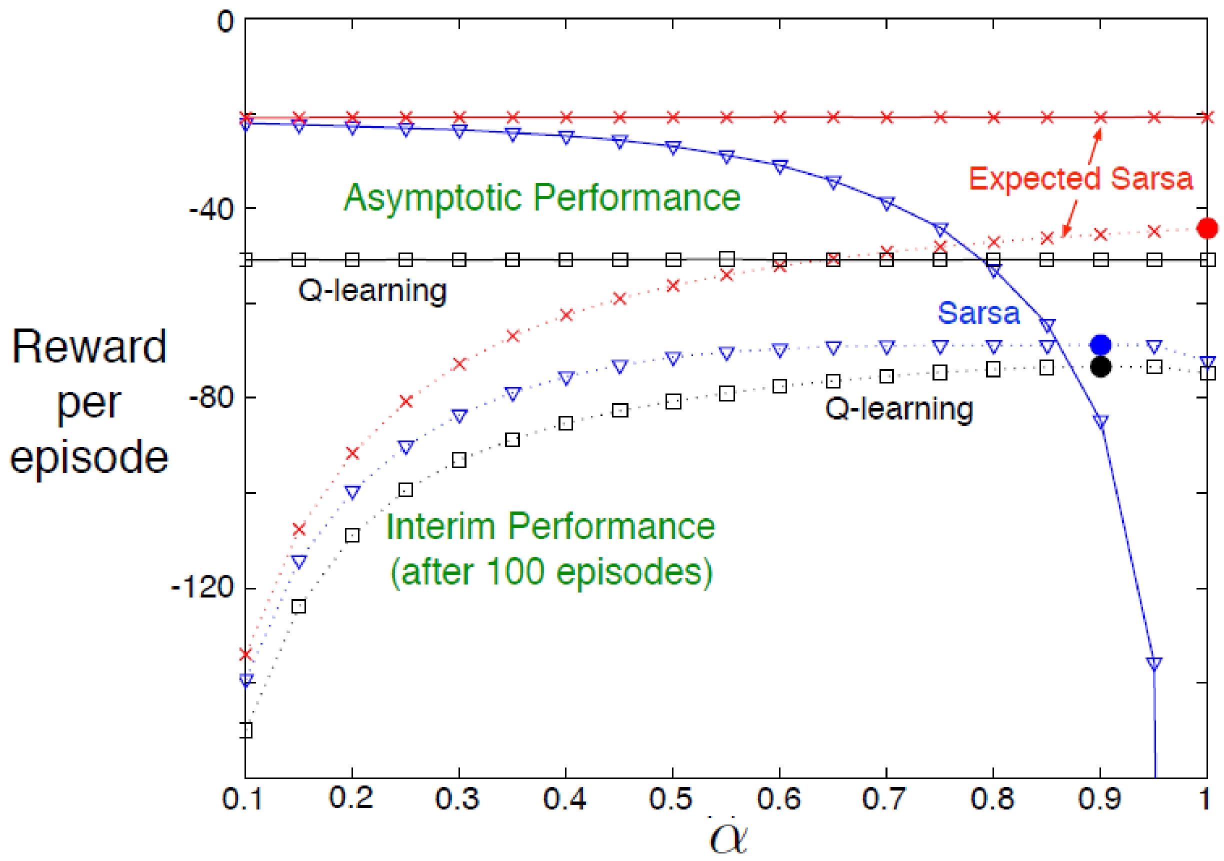




Q-learning?

Expected SARSA?

## Example 5.1: Gridworld over a cliff



Interim (dashed) and asymptotic (solid) performance for $\varepsilon = 0.1$. Asymptotic performance is obtained for 100.000 episodes.

From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

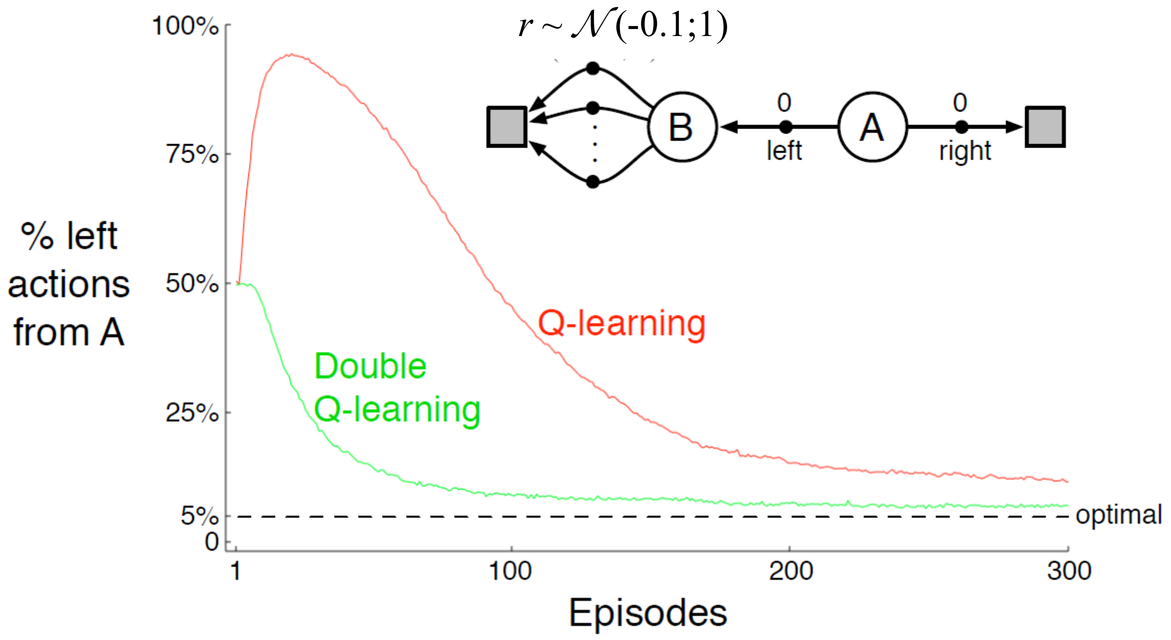
## Maximization bias and double Q-learning

- Bias is due to using the max in the update, either in determining the maximizing action (in SARSA) or in estimating Q (in Q-learning), that is because of using max of estimates as an estimate of the max.
- An example for a simple Markov chain:

$\varepsilon = 0.1$

$\gamma = 1$

$\alpha = 0.1$



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

## Maximization bias and double Q-learning

How to avoid maximization bias?

Problem comes from using the same samples to estimate the function and the maximizing action. Then…

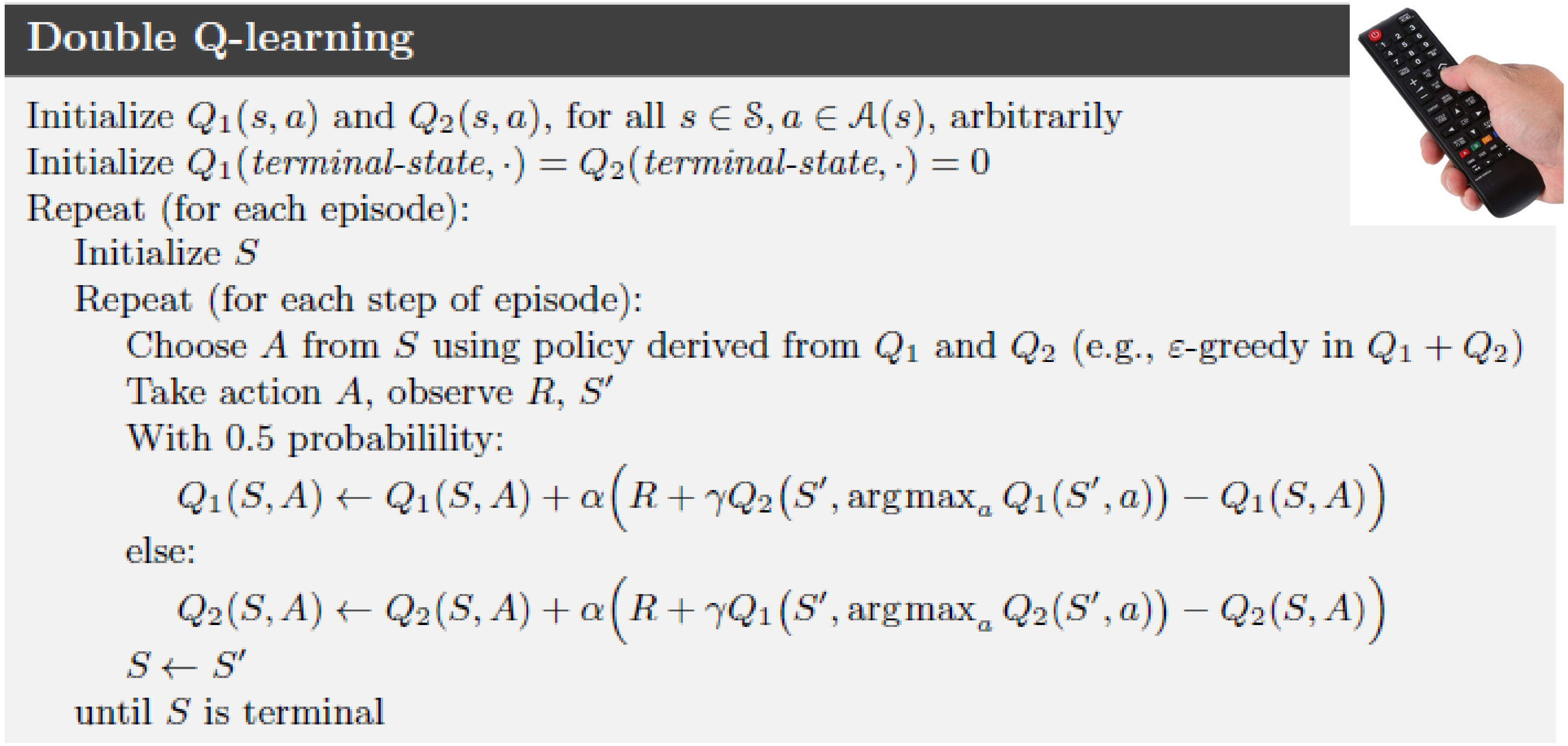
- Suppose we divide the plays in two sets and used them to learn two independent estimates ($Q_1(s,a)$ and $Q_2(s,a)$) each estimate the true value for all $a$ in $\mathcal{A}$.

- Use one estimate, say $Q_1(s,a)$, to determine the action $a^*=\text{argmax}_a\, Q_1(s,a)$, and the other, $Q_2(s,a)$, to provide the estimate of its value, $Q_2(s,a) = Q_2(s, \text{argmax}_a\, Q_1(s,a))$.

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

- This estimate will be unbiased in the sense that $E\{Q_2(s,a)\} = q(s,a)$. We can repeat the process with the role of the two estimates reversed to yield a second unbiased estimate $Q_1(\arg\max_a Q_2(a))$.

# Double Q-learning



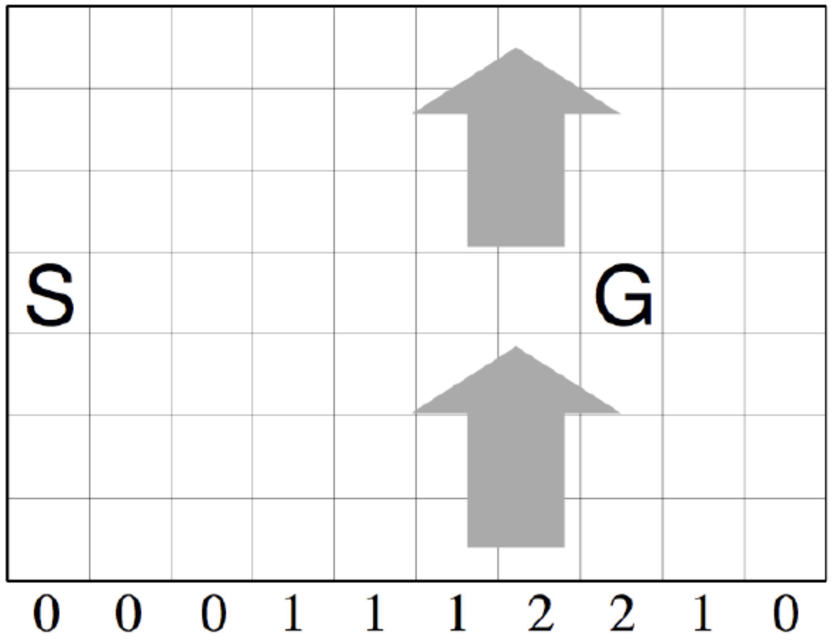
**Double Q-learning**

```
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q_1$ and $Q_2$ (e.g., $\varepsilon$-greedy in $Q_1 + Q_2$)
        Take action $A$, observe $R$, $S'$
        With 0.5 probabilility:
            $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
        else:
            $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
        $S \leftarrow S'$
    until $S$ is terminal
```

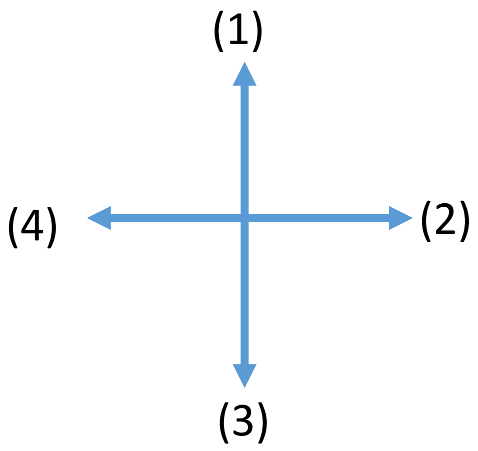
Double versions of Q-learning, SARSA and Expected SARSA do exist.

## Example 5.2: TD for Windy Gridworld (I)

- Standard gridworld
- Start state S and goal state G
- There is a **crosswind upward** through the middle of the grid.
- The **strength of the wind** is given below each column.
- Constant rewards of -1 until the goal state is reached.



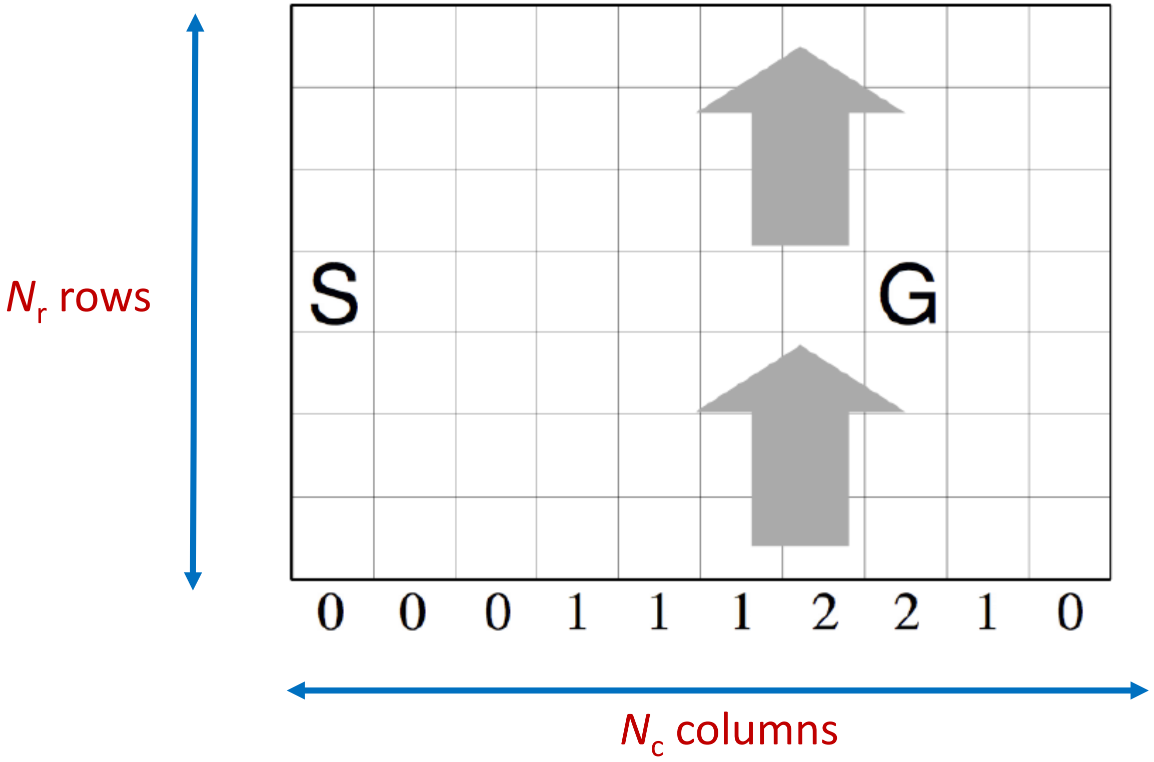
**Four actions**



From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

## Example 5.2: TD for Windy Gridworld (II)

- We have a DMP with $N_s = N_c N_r$ states
- Each episode starts at S and ends at G

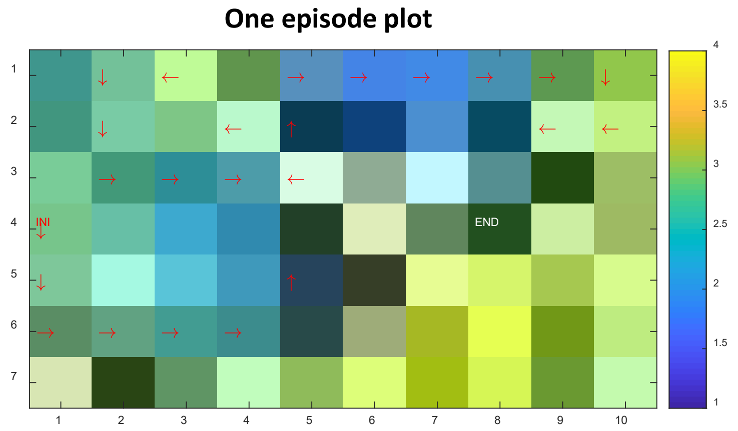


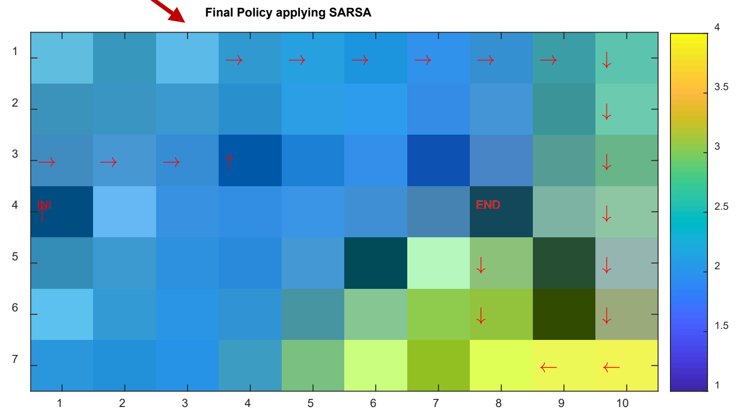
## Example 5.2: TD for Windy Gridworld (III)

SARSA (on-policy control) for estimating $Q = q^*$

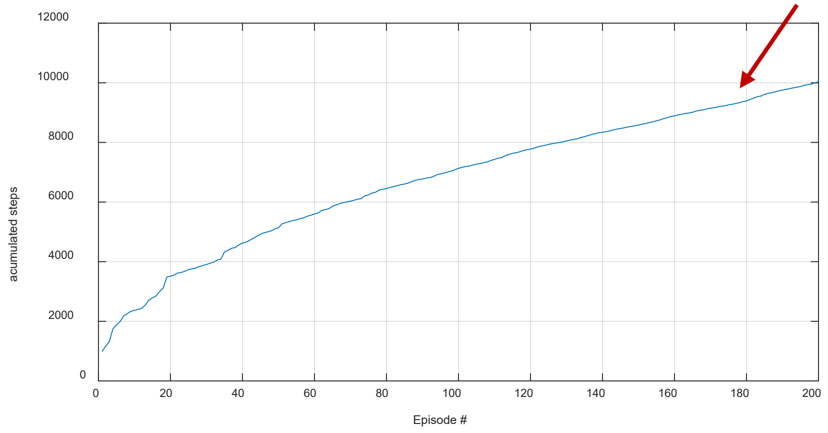
$\gamma = 1,\ \alpha = 0.5,\ \varepsilon = 0.1$

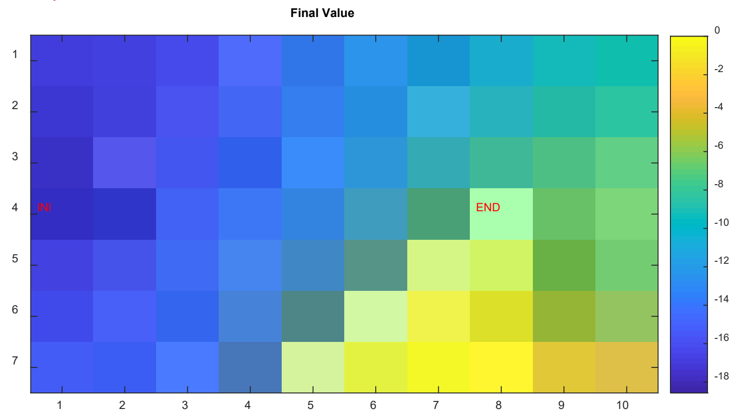
It has not converged





When the optimum policy is obtained the number of steps is fixed





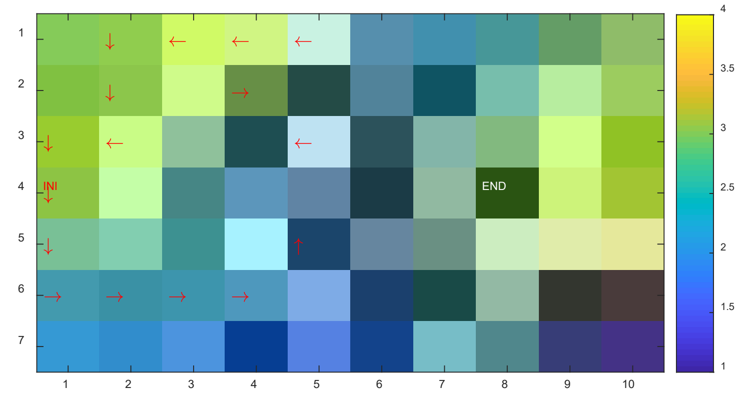
# Example 5.2: TD for Windy Gridworld (IV)

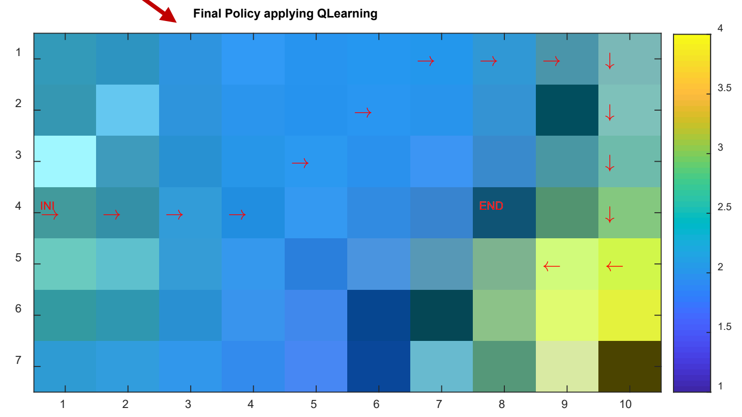
Q-Learning (off-policy control) for estimating $Q = q^*$

$\gamma = 1,\ \alpha = 0.5,\ \varepsilon = 0.1$

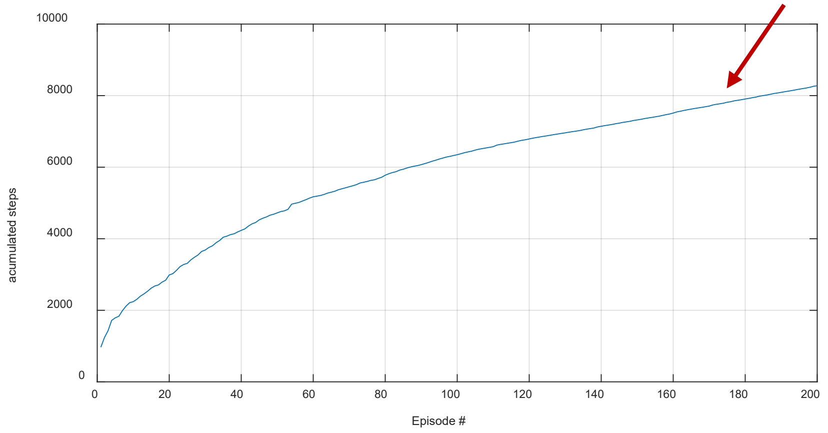
It has converged

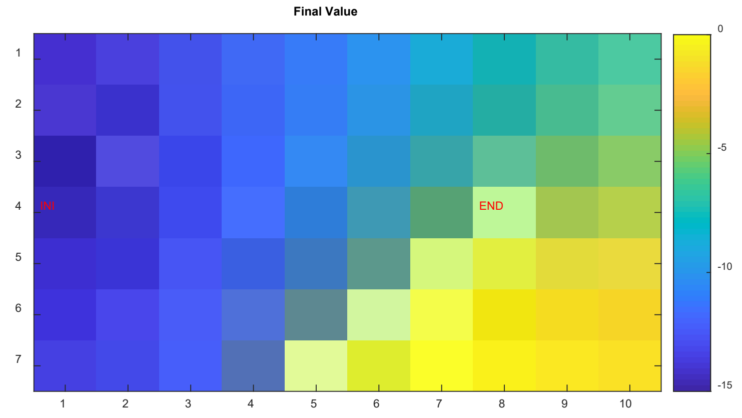
One episode plot





When the optimum policy is obtained the number of steps is fixed





## Example 5.2: TD for Windy Gridworld (V)

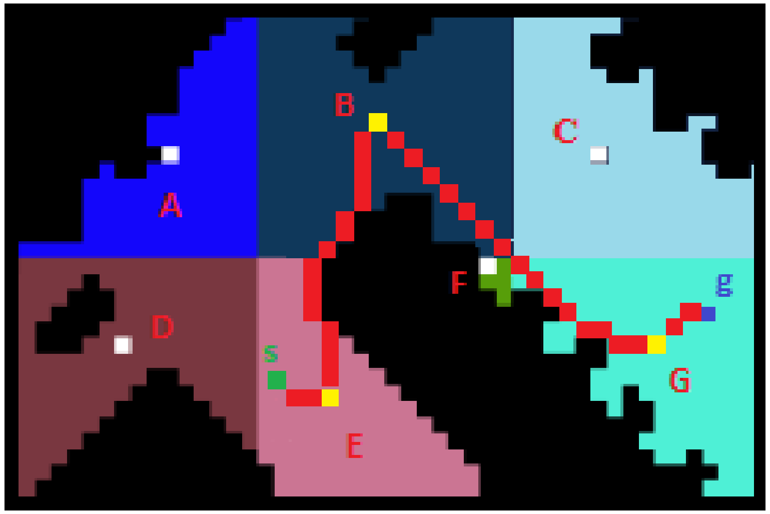
- Q-learning presents a faster convergence to the optimum trajectory.
- Results presented are random, 200 episodes in each case.
- Averaged results over runs should be obtained to get consistent conclusions.

### Applications

- Design of gridworld videogames.
- Options for models:

  - Add diagonal moves, to the usual four

  - Assume stochastic wind strength



## Summary

- As in MC, the control problem is solved using generalized policy iteration (GPI): approximate policy and value functions interact in such a way that both move toward their optimal values.
- One of the two iterations drives the value function to accurately predict returns for the current policy (prediction problem). The other process drives the policy to improve locally.
- A complication arises concerning maintaining sufficient exploration:
  - On-policy: SARSA, Expected SARSA and Double Q-learning
  - Off-policy: Q-learning and Expected SARSA
- The special cases of TD methods introduced in the present chapter should rightly be called one-step, tabular, model-free TD methods.

Quiz #3