Reinforcement learning

Dr. Aissa Boulmerka

The National School of Artificial Intelligence
aissa.boulmerka@ensia.edu.dz

2024-2025

CHAPTER 6
TEMPORAL-DIFFERENCE LEARNING

Outline

- Introduction
- TD Prediction
- Sarsa: On-policy TD Control
- Q-learning: Off-policy TD Control
- Expected Sarsa

Introduction

Introduction

- **Temporal-difference** (TD) learning is a **central and novel idea** in reinforcement learning.
- TD learning is a combination of **Monte Carlo** ideas and **dynamic programming** (DP) ideas.
- Like Monte Carlo methods, TD methods can learn directly from **raw experience** without a **model** of the environment’s dynamics.
- Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they **bootstrap**).
- The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of **reinforcement learning**.

TP Prediction

Review : Estimating Values from Returns

- Recall that in the prediction problem, our goal is to learn a **value function** that estimates the **returns** starting from a given state.

- A simple **every-visit Monte Carlo method** suitable for non-stationary environments is

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s]$$
$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- Bootstraping:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma G_{t+1}$$
$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned}$$

- Monte Carlo methods must wait until the **end of the episode** to determine the increment to $V(S_t)$ (only then is G_t known).

Temporal-Difference

- At time $t + 1$ **Temporal Difference** (TD) methods immediately form a **target** and make a useful update using the observed **reward** R_{t+1} and the **estimate** $V(S_{t+1})$.

- The simplest TD method makes the update immediately on **transition** to S_{t+1} and receiving **reward** R_{t+1}.

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

- The **target for the Monte Carlo update** is $\boldsymbol{G_t}$, whereas the **target for the TD update** is $\boldsymbol{R_{t+1} + \gamma G_{t+1}}$.

- This TD method is called **TD(0)**, or ***one-step* TD**, because it is a special case of the TD(λ) and *n-step* TD methods.

- TD methods combine the **sampling** of MC with the **bootstrapping** of DP.

TD error

- The quantity in brackets in the **TD(0)** update is a sort of **error**, measuring the difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma G_{t+1}$.
- This quantity, called the **TD error**, arises in various forms throughout reinforcement learning:

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

Exercise:

Show that the Monte Carlo error can be written as a sum of TD errors as:

$$\boldsymbol{G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k}$$

TD Prediction

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 $V(\textit{terminal}) = 0$

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'

        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$

        $S \leftarrow S'$

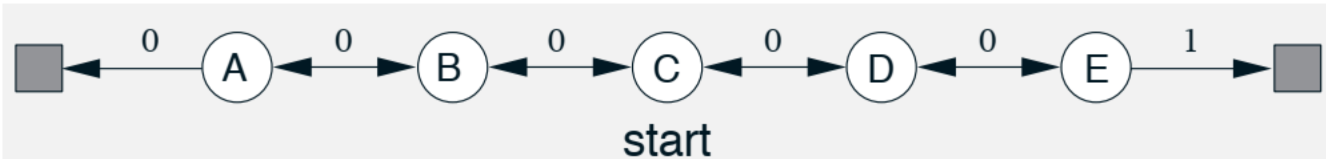
    until S is terminal

Advantages of TD Prediction Methods

- TD methods have an advantage over **DP methods** in that they do not require a **model** of the environment, of its reward and next-state probability distributions.
- The next most obvious advantage of TD methods over MC methods is that they are naturally implemented in an **online**, fully **incremental** fashion, especially for **very long episodes** and **continuing tasks**.
- TD methods converges **faster** than MC methods.

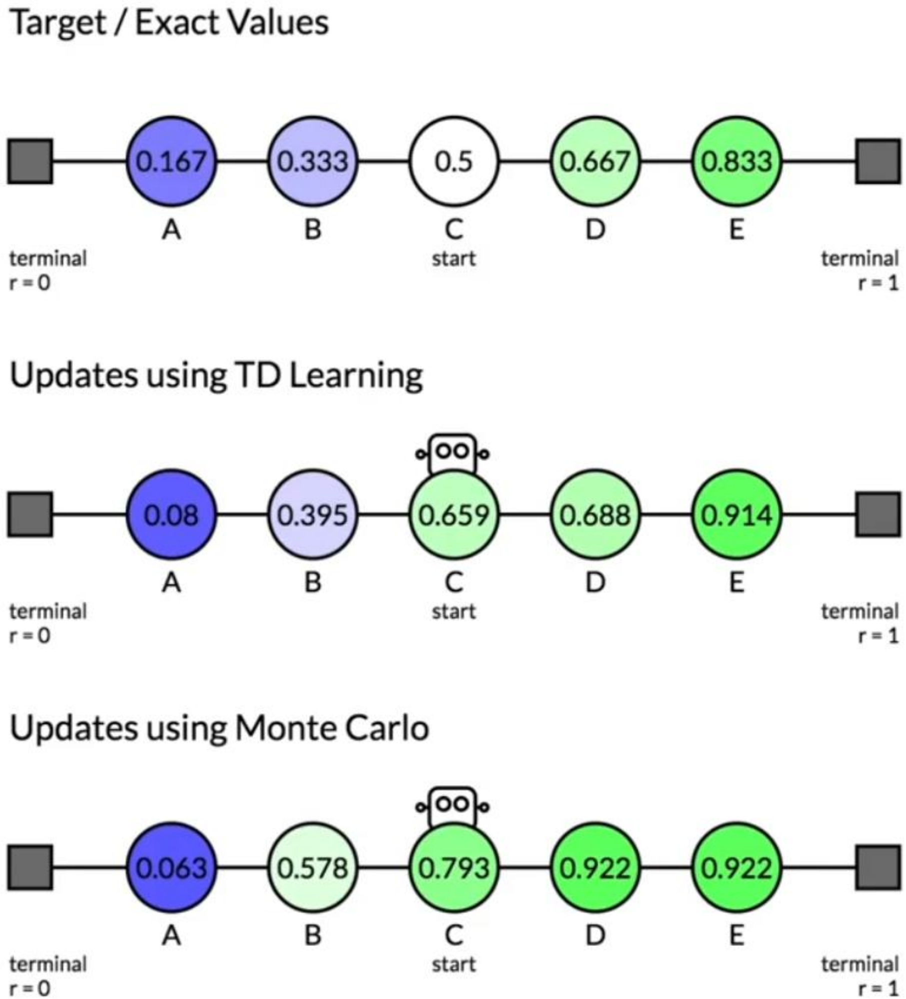
Example: Random Walk

- In this example we empirically compare the prediction abilities of **TD(0)** and **constant-α MC** when applied to the following MDP:



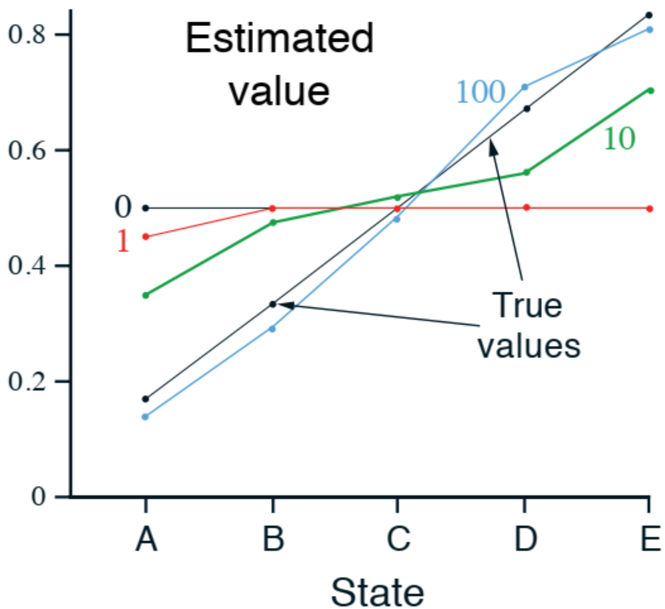
- All episodes start in the **center state, C,** then proceed either **left (←)** or **right (→)** by one state on each step, with equal probability $\pi(.|s) = 0.5, \forall\, s \in \mathcal{S}$.
- Episodes **terminate** either on the **extreme left** or the **extreme right**.
- When an episode terminates on the **right**, a **reward of +1** occurs; all other **rewards are zero**.
- The **discount factor** to be one $\boldsymbol{\gamma = 1}$.

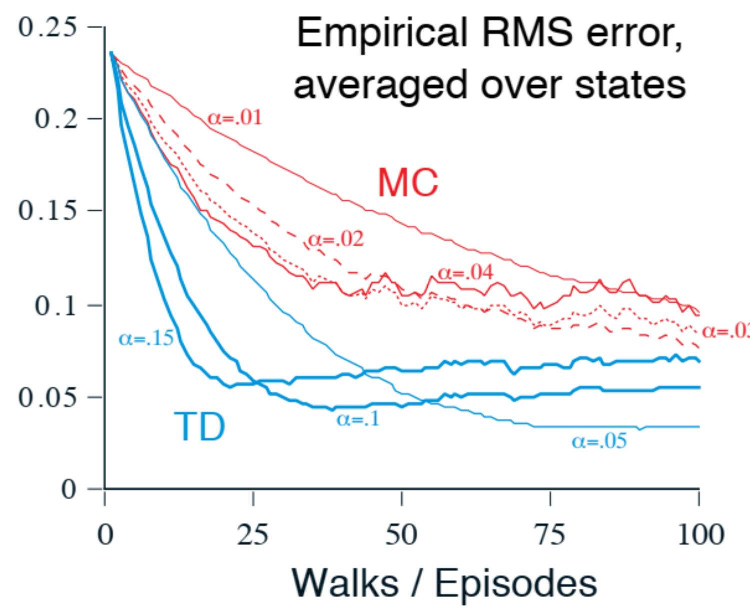
Example: Random Walk



Example: Random Walk

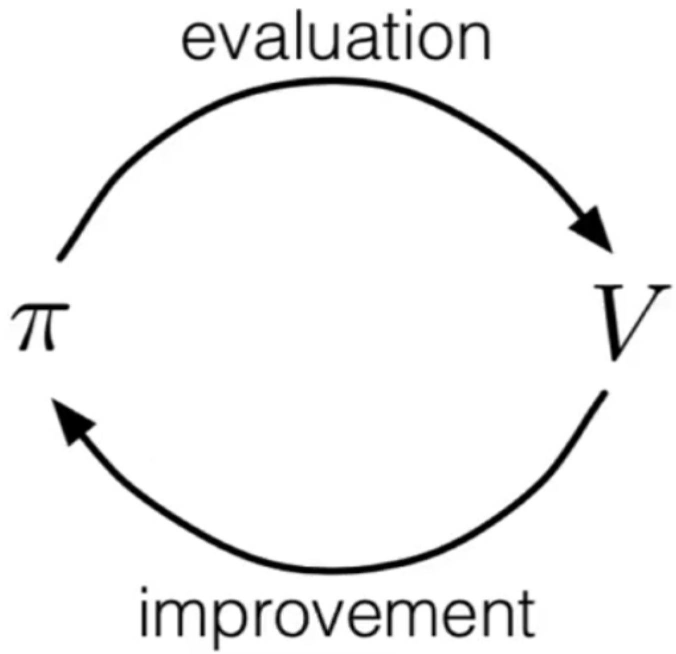
- The left graph above shows the values learned after **various numbers of episodes** on a single run of TD(0).
- The right graph shows learning curves for the two methods for various values of α. The **TD method** was **consistently better** than the **MC method** on this task.





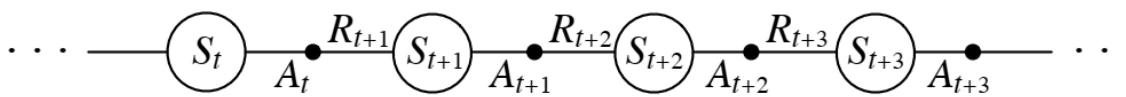
SARSA: On-Policy TD Control

Recall Generalized Policy Iteration



Sarsa: On-policy TD Control

- The first step is to learn an **action-value** function rather than a **state-value function**.
- In particular, for an **on-policy method** we must estimate $\boldsymbol{q_\pi(s,a)}$ for the current **behavior policy** $\boldsymbol{\pi}$ and for all states s and actions a.
- Recall that an episode consists of an alternating sequence of states and state-action pairs:



- In the previous section we considered transitions from **state to state** and learned the **values of states**. Now we consider transitions from **state–action pair** to **state–action pair**, and learn the **values of state–action pairs**.
- Formally these cases are identical: they are both **Markov chains** with a **reward process**.

Sarsa: On-policy TD Control

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

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

- This update is done after every **transition** from a **nonterminal state** S_t. If S_{t+1} is **terminal**, then $\gamma Q(S_{t+1}, A_{t+1})$ is **defined as zero**.
- This rule uses every element of the **quintuple** of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ that make up a transition from one **state–action pair** to the next. This quintuple gives rise to the name **SARSA** for the algorithm.
- It is straightforward to design an **on-policy control algorithm** based on the **Sarsa prediction method**.
- We continually **estimate q_π** for the **behavior policy π**, and at the same time **change π** toward greediness **with respect to q_π**.

Sarsa: On-policy TD Control

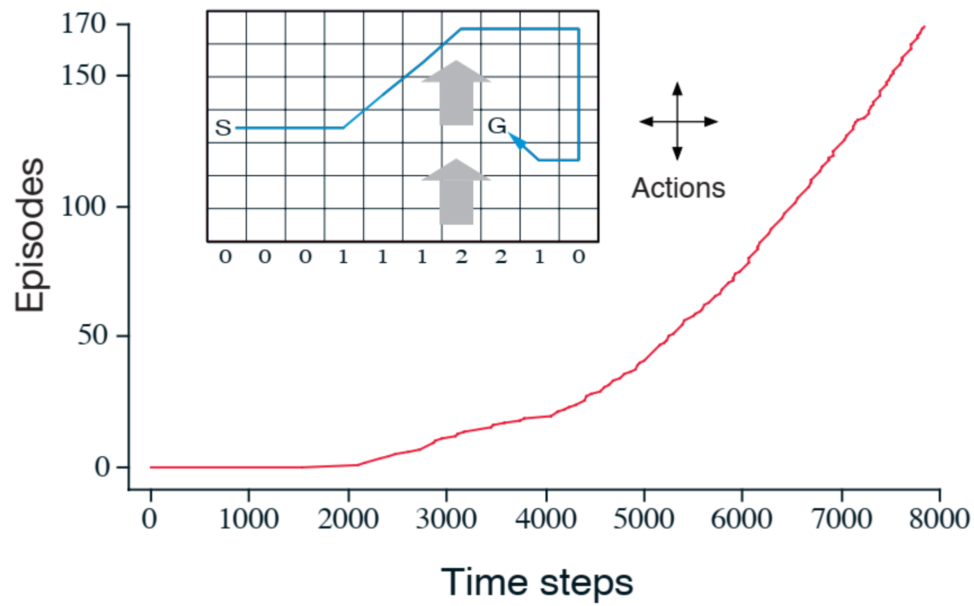
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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 $Q(\mathit{terminal}, \cdot) = 0$

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

Example: Windy Gridworld

- This is a standard gridworld, with **start and goal states**, but with one difference: there is a **crosswind** running upward through the middle of the grid.
- The actions are the standard four: **up, down, right, and left**, but in the middle region the resultant next states are shifted upward by a “wind,” the strength of which varies from column to column.
- The **strength of the wind** is given below each column, in number of cells shifted upward.



Example: Windy Gridworld

- The following graph shows the results of applying **ε-greedy Sarsa** to this task, with $\varepsilon = 0.1, \alpha = 0.5$, and the initial values $Q(s, a) = 0$ for all s, a.
- The increasing slope of the graph shows that the goal was reached **more quickly over time**.
- Note that **MC methods** cannot easily be used here **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**.
- Online learning methods such as **Sarsa** do not have this problem because they **quickly learn** during the episode that such **policies are poor**, and switch to something else.

Q-Learning: Off-Policy TD Control

Q-learning: Off-policy TD Control

- One of the early breakthroughs in reinforcement learning was the development of an **off-policy TD control algorithm** known as **Q-learning**, defined by

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

- In this case, the learned **action-value function $\boldsymbol{Q}$**, directly **approximates $\boldsymbol{q_*}$**, the **optimal action-value function**, independent of the policy being followed.

- This dramatically simplifies the **analysis of the algorithm** and enabled early **convergence proofs**.

- The **policy** still has an effect in that it determines which **state–action pairs** are **visited and updated**. However, all that is required for **correct convergence** is that all pairs continue to be updated.

- Q has been shown to **converge** with probability 1 to q_*.

Q-learning: Off-policy TD Control

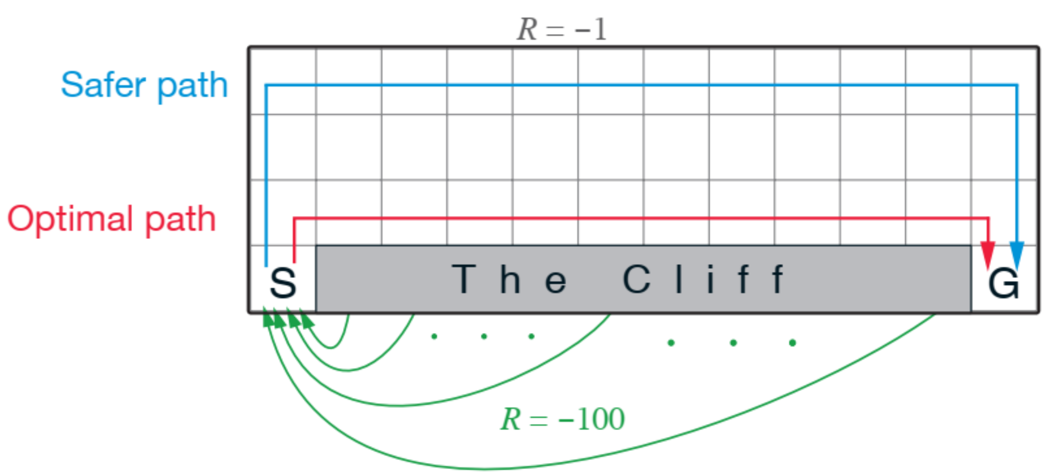
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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 $Q(terminal, \cdot) = 0$

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

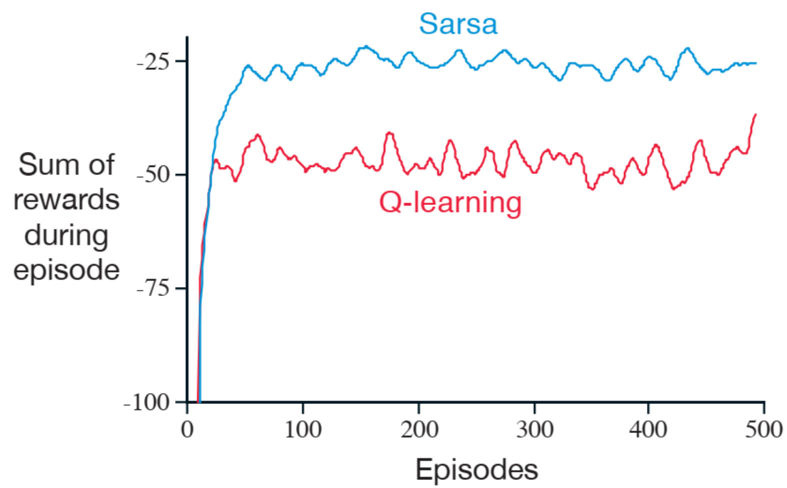
Example: Cliff Walking

- This gridworld example compares **Sarsa** and **Q-learning** highlighting the difference between **on-policy (Sarsa)** and **off-policy (Q-learning)** methods.
- This is a standard **undiscounted**, **episodic task**, with **start** and **goal** states, and **actions** = {up, down, right, and left}. **Reward is -1** on all transitions except those in the **Cliff region**. Stepping into this region incurs a **reward of -100** and sends the agent instantly back to the start.



Example: Cliff Walking

- With **ε-greedy** action selection, $\varepsilon = 0.1$, after an initial transient, Q-learning learns values for the **optimal policy**, that which travels **right along the edge of the cliff**.
- Unfortunately, this results in its occasionally **falling off the cliff** because of the ε-greedy action selection.



- Sarsa, on the other hand, takes the **action selection** into account and learns the **longer but safer path** through the **upper part of the grid**.
- Although Q-learning actually learns the **values of the optimal policy**, its online performance is worse than that of Sarsa, which learns the **roundabout policy**. Of course, if ε were **gradually reduced**, then both methods would **asymptotically converge** to the optimal policy.

Expected SARASA

Expected Sarsa

- The **expected Sarsa** algorithm 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.

- The **expected Sarsa** algorithm uses the following **update rule**:

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

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

Expected Sarsa

Expected Sarsa (on-policy TD control) for estimating $Q \approx \mathbf{q}_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$ arbitrarily for all states s and actions a, except that $Q(terminal, .) = 0$

Initialize policy π based on Q (e.g., ε-greedy)

For each episode:

  Initialize state s

  While s is not terminal:

    Choose action a using policy π (e.g., ε-greedy)

    Take action a, observe reward R and next state s'

    Update $Q(s, a)$ using Expected Sarsa update rule:

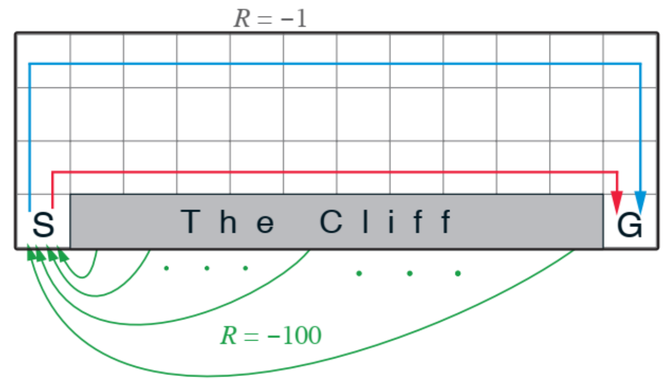
$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[R + \gamma \sum_{a'} \pi(s'|a') \times Q(s', a') - Q(s, a)\right]$$

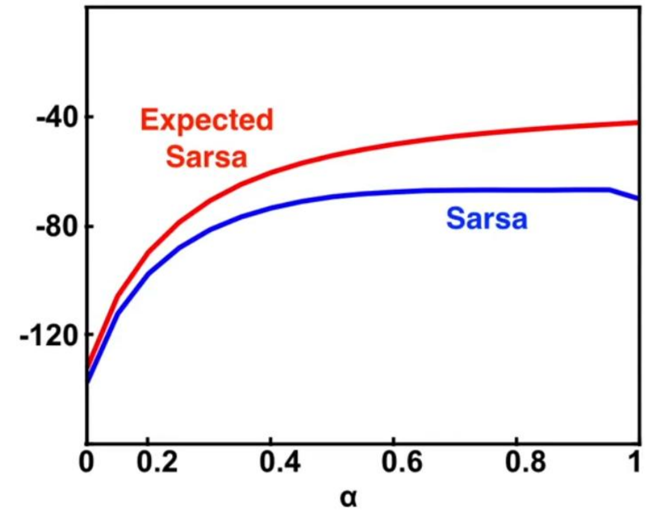
    $s \leftarrow s'$

Return Q and π

Expected Sarsa: Cliff walking environment

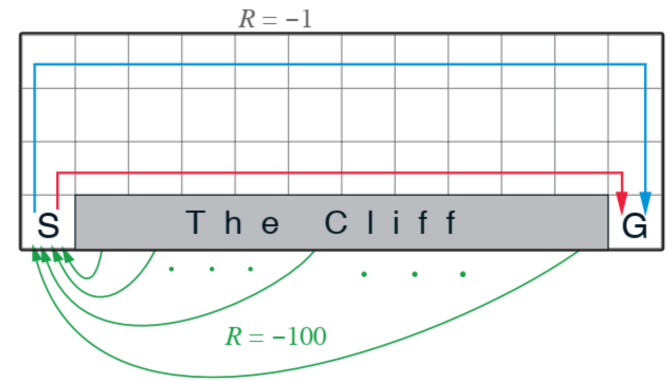
- Hundred episodes and average everything over 50000 independent runs.
- Expected Sarsa **outperformed** Sarsa for almost all **values of alpha**. Expected Sarsa is able to use larger alpha values **more effectively**. This is because it **explicitly averages** over the randomness due to its own policy.
- This environment is **deterministic**, so there are no other sources of randomness to account for.
- This means expected Sarsa's updates are **deterministic** for a given **state and action**. Sarsa's updates on the other hand can **vary significantly** depending on the **next action**.

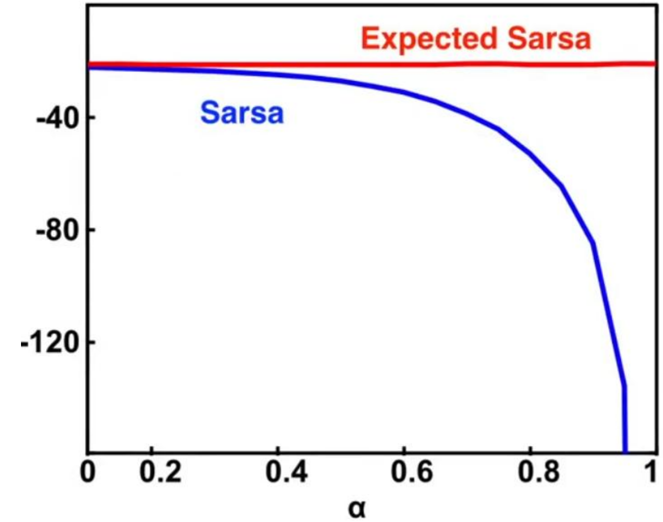




Expected Sarsa: Cliff walking environment

- The **average return per episode** after 100,000 episodes.
- Expected Sarsa's **long-term behavior** is **unaffected** by α. Its updates are deterministic in this example.
- Therefore the **step size** only determines how quickly the estimates approach their target values.
- Sarsa behaves quite **differently** here, it even **fails to converge** for larger values of alpha.
- As **α decreases**, Sarsa's long run performance approaches expected Sarsa's.





Thank you!
Q/A