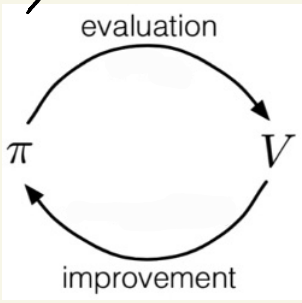
6.4 - Sarsa: On-Policy TD Control:

Jan 4, 2022

Now we want to use TD methods for control.

↳ Still follows pattern of generalized policy iteration (GPI), like w/ MC.



↳ Use an on-policy TD control method.

↳ Estimate action-value functions now (like w/ MC) instead of state-action functions like previously.

State-Action Pair Update Rule:

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

✡ Update is done after every transition from a nonterminal state S_t.

↳ This rule uses a quintuplet of events that make up a transition, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

SARSA

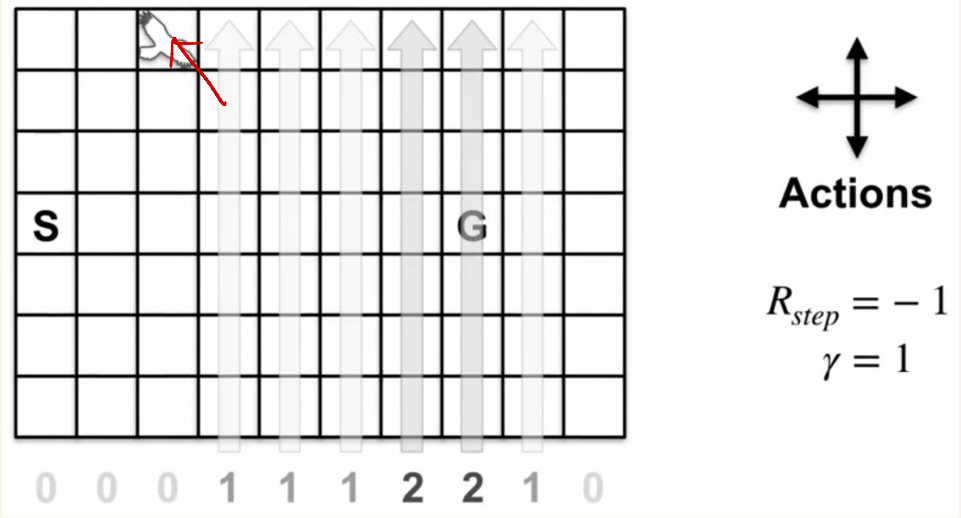
Sarsa Pseudocode!

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

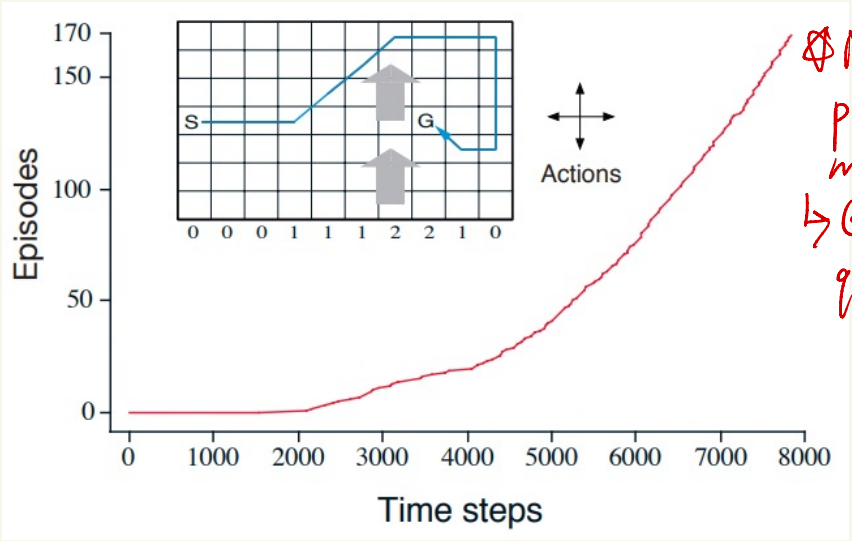
Windy Gridworld Example!



↳ Wind blows agent upwards when moving out of certain states.

↳ Hitting boundaries does nothing.

Sarsa; $\varepsilon = 0.1$, $\alpha = 0.5$



- More episodes passed after more time steps.
  - ↳ Goal reached more quickly over time.

- A deterministic policy might get trapped here.
  - ↳ Episode never ends; an MC method would never learn the optimal policy.

6.5 - Q-Learning: Off-Policy TD Control:

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

↳ Q directly approximates optimal action-value function q_*, independent of the policy followed.

↳ Differs from Sarsa cuz it doesn't use A_{t+1}.

Q-Learning 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

Why Doesn't Q-Learning Use A_{t+1}?

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

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

* Bellman eqn for action values

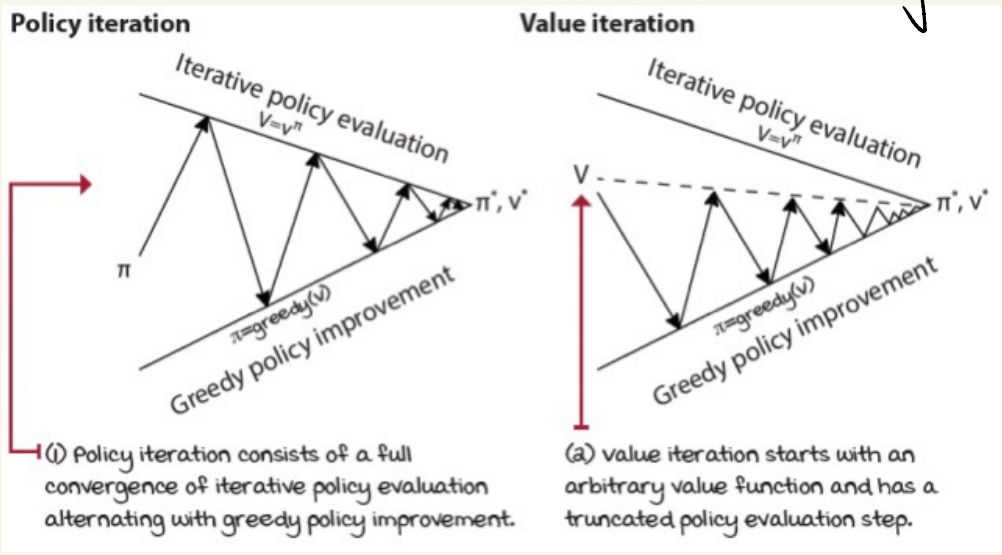
Q-learning: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\right)$

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

* Bellman's Optimality Eqn for action values

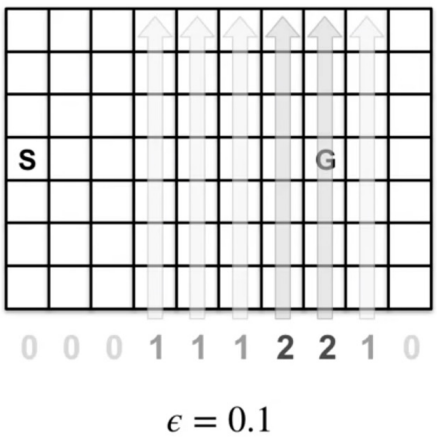
Sarsa

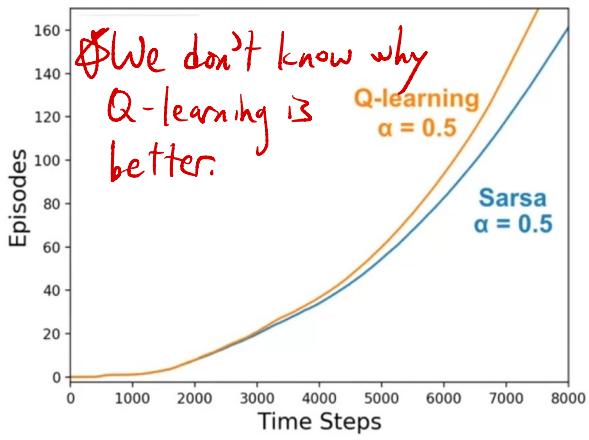
Q-Learning



* Remember that the Bellman Optimality Equation allows Q-Learning to directly learn q_* w/t switching b/t policy improvement and evaluation.
  - Value iteration is just faster.

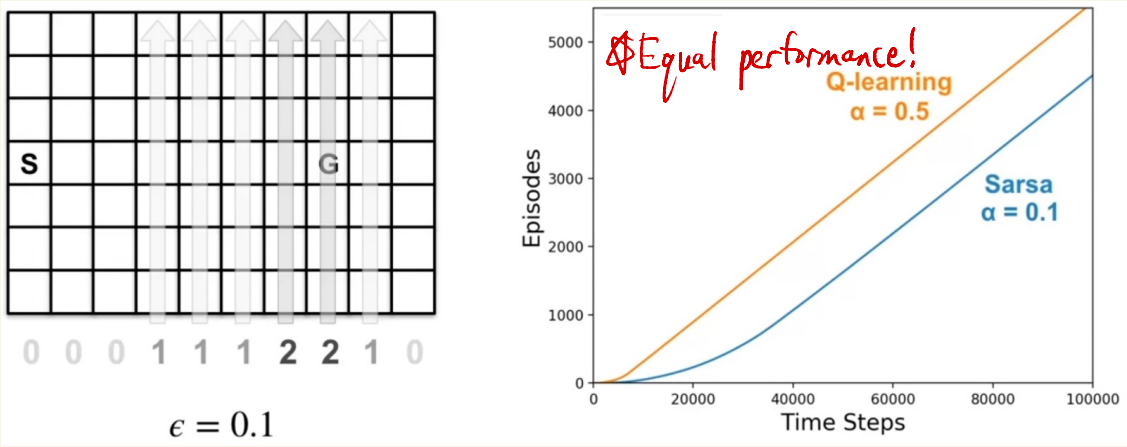
Results for Windy Gridworld:





How do we make Sarsa perform better?

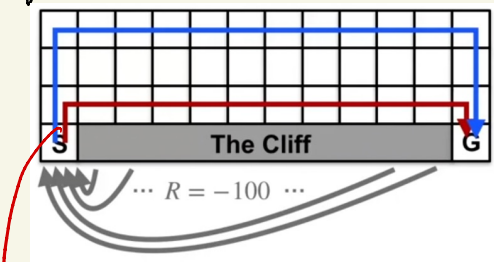
↳ Run for more time steps w/ $\alpha = 0.1$!



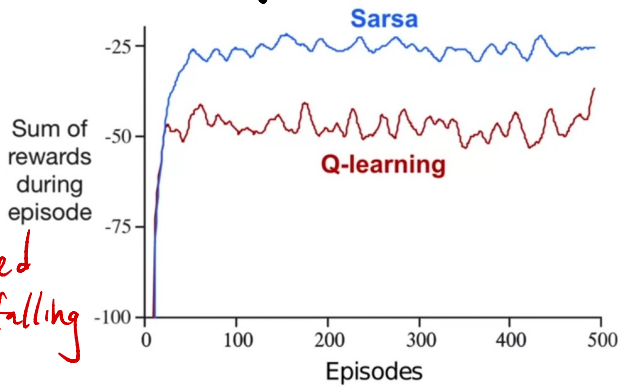
※ Equal performance!

※ Slopes end up the same, meaning both algorithms have converged to the same policy.

However, sometimes Sarsa performs better by default cuz ϵ-greedy performs better than Q-learning's greedy policy.



※ Cliff walking environment



↳ The greedy policy is learned quickly, but more prone to falling off cliff.

6.6 - Expected Sarsa:

Same as Sarsa, but a deterministic algorithm instead of an expectation one.

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

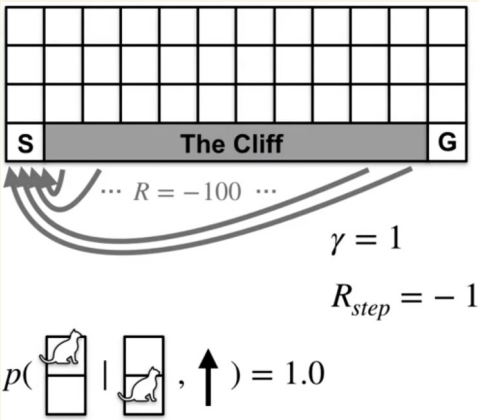
$$\downarrow$$

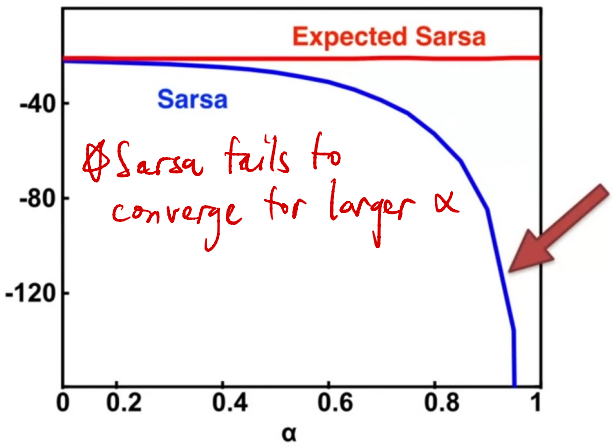
$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \mathbb{E}_\pi\left[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}\right] - Q(S_t, A_t)\right] \\ &= Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\right] \end{aligned}$$

algorithm moves deterministically given next state S_{t+1}

- More computationally expensive, but eliminates variance from random selection of A_{t+1}.

Comparison in Cliff Walking Environment:





- Sarsa fails to converge for larger α

∴ Expected Sarsa is more robust for larger α.

Summary:

✡ Action-value Bellman equation

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

↳ Sarsa: $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ ✡ On-policy

↳ Expected Sarsa: $R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1})\, Q(S_{t+1}, a')$ ✡ On/Off-policy

✡ Bellman optimality equation

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

↳ Q-learning: $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ ✡ Off-policy