The Sarsa Algorithm

$$\begin{array}{cccccc} S & A & R & S & A \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ S_t & A_t & R_{t+1} & S_{t+1} & A_{(t+1)} \end{array}$$

The Sarsa algorithm makes predictions about the values in the state action pairs.

1. The agent chooses the action in the initial state to create the first state-action pair (S_t & A_t).
2. The algorithm takes the action and observes the next reward and next state (R_{t+1} & S_{t+1}).
3. The agent commits its next action before updating the value estimate.

By doing this, Sarsa algorithms allows the agent to update the estimate every step instead of every episode

The full Sarsa update equation:

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

Comparison of Sarsa and Q-learning

Sarsa algorithm is based on Bellman equation

Q-learning algorithm is based on Bellman Optimality equation

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

Algorithm parameters: $\alpha \in (0,1], \epsilon > 0$
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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
Until S is terminal

The expected Sarsa algorithm

The expected Sarsa algorithm explicitly computes the expectation under its policy, which is more expensive than sampling but has lower variance.

Expected Sarsa can learn a good policy faster and is more robust to large step sizes

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$