SARSA Tutorial

September 21, 2025

1 Introduction

SARSA (State-Action-Reward-State-Action) is an on-policy temporal-difference reinforcement learning algorithm. Unlike Q-learning, SARSA updates action values using the action actually taken in the next state, resulting in a policy that incorporates the exploration strategy employed during learning.

2 Theory and Formulas

2.1 On-Policy Action-Value Function

SARSA estimates the action-value function $Q(s, a)$ under the current policy π. The TD target uses the next action selected by π:

$$Q^{\pi}(s, a) = \mathbb{E}\big[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a\big]. \tag{1}$$

2.2 Update Rule

After observing transition $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, SARSA performs

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \Big[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\Big]. \tag{2}$$

The policy used for action selection, often ε-greedy, directly influences the update.

2.3 Convergence Properties

If decreasing learning rates satisfy $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$, and policy exploration ensures all state-action pairs are visited infinitely often, SARSA converges to an optimal ε-greedy policy in finite MDPs. Unlike Q-learning, the learned policy accounts for exploration, making SARSA more conservative near risky states.

3 Applications and Tips

- **Stochastic control**: environments where exploratory actions incur risk, such as navigation with slip probabilities.
- **Robotic tasks**: smooth policies that respect exploration noise (e.g., SARSA with eligibility traces).

- **Education/training simulators**: adapt strategies that internalize exploration effects.
- **Best practices**: tune ε decay to balance exploration, compare returns with Q-learning to assess risk sensitivity, and monitor variance across seeds.

4 Python Practice

The script `gen_sarsa_figures.py` trains SARSA on a stochastic grid-world where moving near cliffs yields large penalties. It visualizes episode returns and final state-value estimates under the learned on-policy strategy.

Listing 1: Excerpt from $\text{gen}_s arsa_f igures.py$

```
for episode in range(num_episodes):
    state = env.reset()
    action = epsilon_greedy(Q[state], epsilon)
    done = False
    G = 0.0
    while not done:
        next_state, reward, done = env.step(state, action)
        next_action = epsilon_greedy(Q[next_state], epsilon)
        td_target = reward + gamma * Q[next_state, next_action] *
           (1.0 - float(done))
        Q[state, action] += alpha * (td_target - Q[state, action])
        state, action = next_state, next_action
        G += reward
    returns.append(G)
```

5 Result

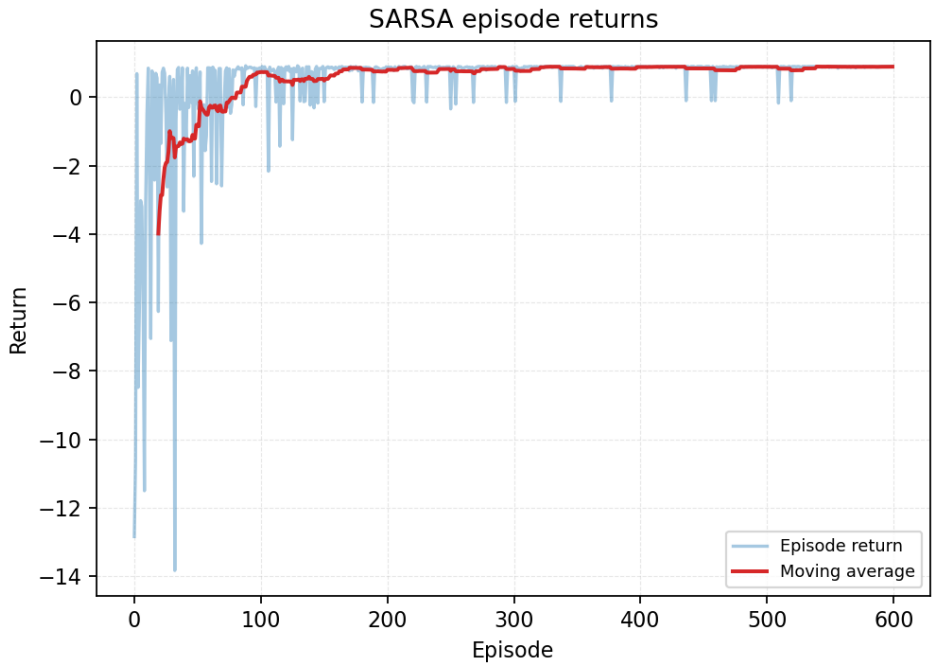

Figure 1: SARSA episode returns showing convergence under ε-greedy policy

Figure 2: State-value heatmap reflecting the conservative policy learned near hazardous states

6 Summary

SARSA integrates exploration behaviour into its updates, yielding on-policy control well-suited for stochastic or risk-sensitive tasks. Proper tuning of learning rate and exploration schedule ensures stable convergence. The grid-world example highlights how returns stabilize and how the resulting value function captures safe routes.