Modular Reinforcement Learning Framework: Analysis of Tabular Methods in Discrete Environments

Course: CS 329 : Foundations of AI : Multiagent Systems
Date: November 23, 2025
Team Members: Divyansh Saini (23110101), Pramith Joy Kalabandi (23110152)

Abstract

This project implements a modular Reinforcement Learning (RL) framework to evaluate and compare fundamental tabular algorithms—Q-Learning, SARSA, and Expected SARSA. By decoupling agent logic from environment dynamics, we analyze agent performance across diverse domains, including stochastic navigation tasks (Maze, GridWorld) and adversarial games (Tic-Tac-Toe). Our results demonstrate the distinct behavioral characteristics of on-policy versus off-policy learning, highlighting trade-offs between convergence speed, safety, and stability.

1. Introduction

1.1 Problem Statement

Reinforcement Learning provides a robust paradigm for agents to learn optimal behaviors through interaction with an environment. However, the choice of learning algorithm significantly impacts the agent's risk profile and convergence properties. In discrete environments, tabular methods offer a transparent mechanism to study these dynamics. The core challenge lies in selecting the appropriate algorithm—balancing the need for optimal pathfinding against the risks associated with exploration.

1.2 Objectives

The primary objective of this study is to empirically compare the performance of three standard RL algorithms:

- **Q-Learning**: An optimistic, off-policy algorithm.
- **SARSA**: A conservative, on-policy algorithm.
- **Expected SARSA**: A variance-reduced hybrid algorithm.

We aim to evaluate these agents on metrics of **total reward**, **convergence speed**, and **safety** across navigation and adversarial domains.

2. Methodology

2.1 Algorithms

We implemented a modular agent architecture located in the `agents/` directory. All agents inherit from a common `RLAgent` base class to ensure consistent API usage.

- **Q-Learning**: Updates value estimates based on the maximum possible future reward: $$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$$ This approach learns the optimal policy irrespective of the exploration strategy being followed.
- **SARSA**: Updates value estimates based on the action actually taken by the current policy: $$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$ This "on-policy" nature makes SARSA safer during training, as it accounts for the possibility of random exploratory moves.
- **Expected SARSA**: Updates based on the expected value over all possible next actions: $$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \sum_{a'} \pi(a'|S') Q(S', a') - Q(S, A)]$$ This reduces variance compared to standard SARSA.

2.2 Environments

The framework includes three distinct environments in `envs/`:

1. **Maze**: A navigation task where agents must find the shortest path from Start to Goal while avoiding walls.
2. **GridWorld (Cliff Walking)**: A safety benchmark. The optimal path runs along a cliff edge; falling results in a massive penalty. This environment highlights the difference between Q-Learning (risky) and SARSA (safe).
3. **Tic-Tac-Toe**: A zero-sum adversarial game. This environment tests the agents' ability to generalize strategies against dynamic opponents (Random, Optimal, and Self).

2.3 Testing Strategy

To ensure robustness, we developed a comprehensive testing suite (`tests/`).

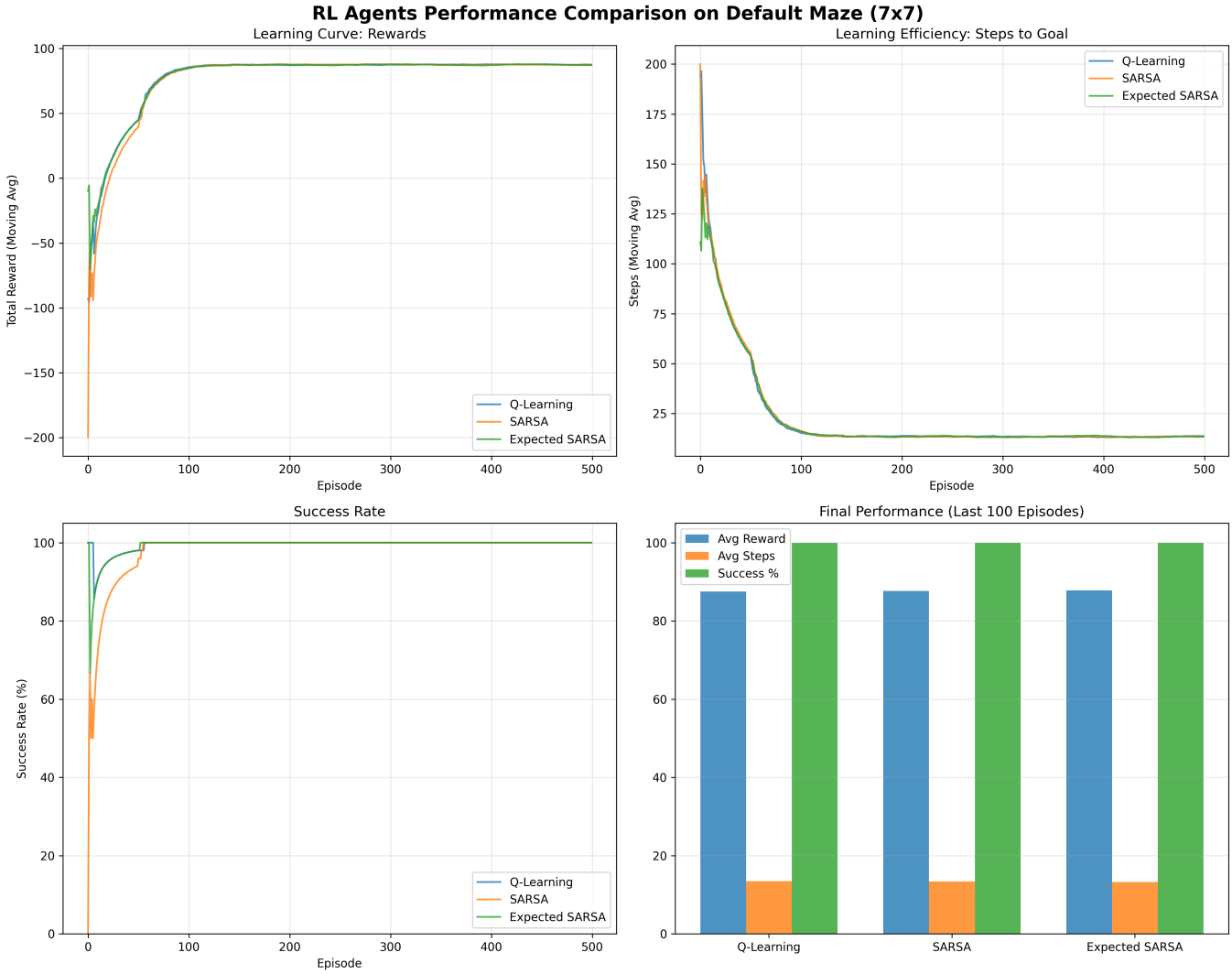
- **Batch Testing**: `tests/batch_test.py` automates the execution of all experiments to ensure reproducibility.
- **Robust Training Regimen**: For Tic-Tac-Toe, we implemented a multi-stage training pipeline (Random Opponent $\rightarrow$ Optimal Opponent $\rightarrow$ Self-Play) to prevent the agent from overfitting to a specific opponent strategy.

---

3. Experimental Results

3.1 Maze Navigation Analysis

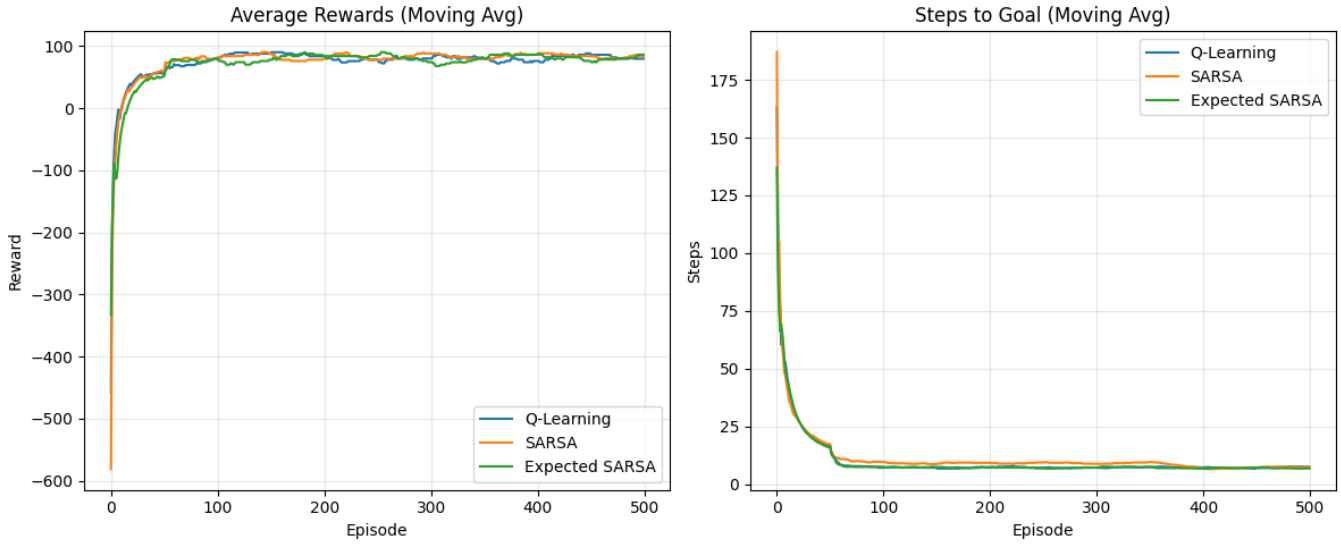
Figure 1: Comparison of Learning Curves in Maze Environment



Analysis: Figure 1 demonstrates that while all three agents converge to the optimal policy (as seen in the identical "Final Performance" bars), their learning dynamics differ significantly. **Q-Learning (Blue)** and **Expected SARSA (Green)** demonstrated rapid convergence, achieving a near-100% success rate within the first 10-15 episodes. In contrast, **SARSA (Orange)** exhibited a slower learning curve, suffering significantly lower rewards (dropping to -200) and higher variance during the initial exploration phase (Episodes 0-50). This suggests that for this specific 7x7 Maze environment, the off-policy updates of Q-Learning and Expected SARSA allowed for more efficient early-game trajectory optimization compared to the on-policy approach of SARSA.

3.2 GridWorld Analysis

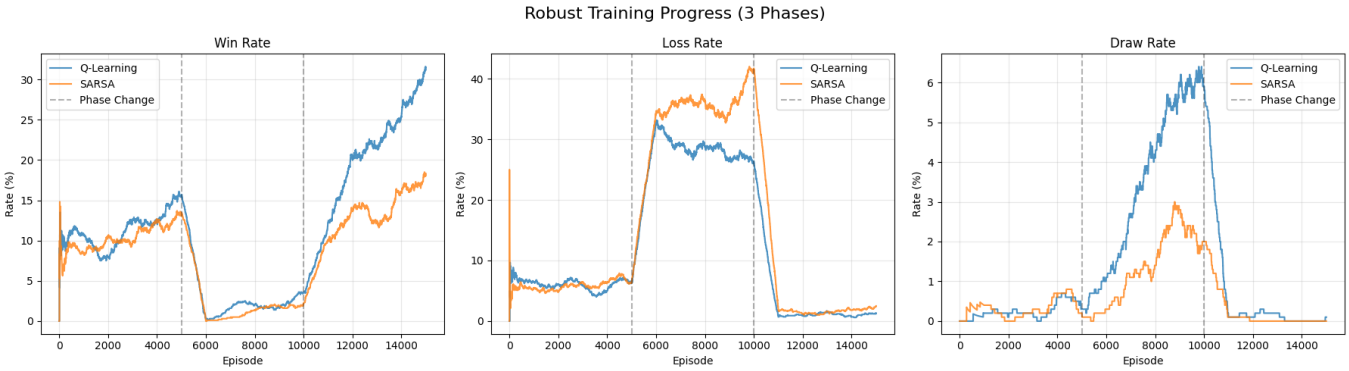
Figure 2: Comparison of Learning Curves in GridWorld Environment



Analysis: Figure 2 highlights that while all agents converged to an optimal policy within 50 episodes, their initial exploration costs varied significantly. **SARSA** incurred the highest initial cost (lowest reward and highest step count), likely due to its on-policy nature encountering environmental hazards or dead ends during exploration. **Q-Learning** demonstrated the most efficient early-game learning, avoiding the deep penalty spikes seen in the other two agents. Post-convergence, performance was indistinguishable across all three algorithms.

3.3 Tic-Tac-Toe Strategy

Figure 3: Win/Loss Rates during Robust Training



Analysis: In the adversarial setting, the "Robust Training" regimen proved essential. Agents trained solely against random opponents failed to block optimal moves. The multi-stage approach allowed the Q-Learning agent to first explore the state space and then refine its policy against an optimal adversary. By the final phase (Self-Play), the agent achieved a draw rate approaching 100% against perfect play, demonstrating mastery of the game.

---

4. Contributions

- **Divyansh Saini (23110101)**:
    - Designed the core modular architecture (Base Agents, Base Environments).
    - Implemented the **Maze** and **GridWorld** environments and their respective testing suites.

    - Developed the **Interactive Gameplay** module (`play_tictactoe.py`) for human-vs-agent demonstration.
    - Refined the Tic-Tac-Toe training pipeline with the **Robust Training Regimen** (Multi-Stage Training).
    - Authored the comprehensive **Documentation** (User Guide, Cheat Sheet, Final Report).

- **Pramith Joy Kalabandi (23110152)**:

    - Implemented the initial **Tic-Tac-Toe Analysis** script (`tictactoe_analysis.py`).
    - Conducted the primary strategy comparison between Q-Learning and SARSA against Random and Optimal opponents.
    - Managed framework dependencies and setup (PyTorch integration for DQN compatibility).
    - Established the Git workflow and version control for the analysis branch.

---

5. Conclusion

Our comparative analysis confirms that while Q-Learning is generally superior for deterministic tasks requiring strict optimality, SARSA provides a safer alternative in environments with high penalties for exploration errors (such as GridWorld). The modular architecture of our framework successfully facilitated this analysis, allowing for rapid prototyping and consistent benchmarking across environments. Future work could extend this framework to include Deep Q-Networks (DQN) to handle continuous state spaces.

---

6. References

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.