Web-Based Interactive Reinforcement Learning Learning Tool

1 Introduction

Reinforcement Learning (RL) is a fundamental area of machine learning in which an agent learns optimal behavior by interacting with an environment and receiving feedback in the form of rewards. Despite its growing importance in modern artificial intelligence systems, RL remains challenging for learners due to its abstract mathematical foundations and limited intuitive visualization.

This project introduces a **web-based interactive reinforcement learning platform** designed to enhance understanding through direct interaction. The system allows users to select different environments, apply various reinforcement learning algorithms, adjust learning parameters in real time, and visualize both training and inference processes. The goal of this tool is to bridge the gap between theoretical RL concepts and practical understanding.

2 System Architecture

The platform follows a client–server architecture consisting of a backend responsible for reinforcement learning computation and a frontend dedicated to visualization and user interaction.

2.1 Backend

The backend is implemented in Python using an asynchronous web framework. Its responsibilities include:

- Managing reinforcement learning environments
- Executing RL algorithms
- Updating learning parameters dynamically
- Streaming environment states and metrics in real time

2.2 Frontend

The frontend is implemented using HTML, CSS, and JavaScript, utilizing the Canvas API for visualization. It provides:

- Interactive controls for algorithm and environment selection
- Parameter tuning interfaces
- Real-time visualization of agent behavior
- Performance plots and policy visualization

Communication between frontend and backend is achieved through WebSockets, enabling low-latency real-time updates.

3 Reinforcement Learning Environments

The platform includes multiple environments designed to demonstrate different reinforcement learning challenges.

3.1 GridWorld

GridWorld is a discrete, tabular environment in which the agent navigates a grid containing obstacles, a start state, and a goal state. It is primarily used for demonstrating dynamic programming and tabular RL algorithms. Visualization includes agent movement, goal location, walls, policy arrows, and value-function heatmaps.

3.2 FrozenLake

FrozenLake is a stochastic grid-based environment consisting of safe tiles, holes, and a goal tile. The environment highlights uncertainty and delayed rewards. Visualizations include agent trajectories, policy arrows, and state-value heatmaps.

3.3 CartPole

CartPole is a continuous control environment where the agent must balance a pole on a moving cart. The visualization displays real-time cart movement and pole angle, illustrating continuous state control challenges.

3.4 MountainCar

MountainCar is a continuous environment in which the agent must drive a car up a steep hill by building momentum. Visualization includes the terrain profile, car position, and goal location, emphasizing sparse rewards and delayed success.

3.5 Breakout

Breakout is a custom arcade-style environment where the agent controls a paddle to break bricks using a bouncing ball. The environment demonstrates complex interaction dynamics and high-dimensional state behavior.

3.6 Gym4ReaL

Gym4ReaL is a custom two-dimensional navigation environment with obstacles and goal regions. It is designed to test spatial reasoning and generalization, with visualization showing agent movement and orientation.

4 Implemented Reinforcement Learning Algorithms

The platform supports a comprehensive set of reinforcement learning algorithms across multiple categories.

4.1 Policy Evaluation

Policy Evaluation computes the state-value function for a fixed policy using full environment dynamics. The resulting value function is visualized using heatmaps.

4.2 Policy Iteration

Policy Iteration alternates between policy evaluation and policy improvement steps to converge to an optimal policy. Learned policies are visualized using directional arrows.

4.3 Value Iteration

Value Iteration combines evaluation and improvement into a single update using the Bellman optimality equation. Convergence behavior is shown through evolving value heatmaps.

4.4 Monte Carlo Methods

Monte Carlo methods learn from complete episodes without requiring a model of the environment. Performance is visualized through episodic return plots.

4.5 Temporal Difference Learning

Temporal Difference (TD) learning updates value estimates incrementally at each time step. Both TD(0) and n-step TD variants are supported, with adjustable step size.

4.6 SARSA

SARSA is an on-policy control algorithm that learns action-value functions while following the current exploration policy. Q-values are visualized using state-action heatmaps.

4.7 Q-Learning

Q-Learning is an off-policy control algorithm that learns the optimal action-value function independently of the agent's behavior. Learned policies are visualized using maximum Q-value heatmaps.

5 Parameter Adjustment Capabilities

A key feature of the platform is real-time parameter adjustment during training. Supported parameters include:

- Discount factor (γ)
- Learning rate (α)
- Exploration rate (ϵ)
- Epsilon decay
- Number of steps in n-step TD
- Maximum episode length
- Reward shaping options
- Random seed for reproducibility

All parameter changes are immediately applied without restarting the training process.

6 Visualization Techniques

6.1 Environment Rendering

Each environment is rendered in real time using canvas-based graphics, allowing users to observe agent movement and interactions.

6.2 Policy Visualization

Policies are visualized using directional arrows in grid-based environments, providing insight into learned behavior.

6.3 Value Function Heatmaps

State-value and action-value functions are visualized as heatmaps, illustrating convergence and exploration patterns.

6.4 Training Performance Plots

Live plots display episodic rewards or scores, enabling users to monitor learning progress and stability.

6.5 Inference Visualization

Inference mode runs the learned greedy policy without exploration, showing the final learned behavior and performance metrics.

7 User Interface Design

The user interface is designed to be intuitive and educational. It includes dropdown menus for environment and algorithm selection, sliders for parameter tuning, control buttons for training and inference, and real-time feedback panels. Incompatible algorithm-environment combinations are automatically handled to prevent execution errors.

8 Conclusion

This project presents a comprehensive web-based reinforcement learning learning tool that enhances understanding through interaction and visualization. By integrating multiple environments, algorithms, real-time parameter tuning, and rich visual feedback, the platform effectively bridges theory and practice. The system provides a strong foundation

for future extensions such as deep reinforcement learning, multi-agent environments, and curriculum-based training.