Assignment: Comparing Reinforcement Learning Algorithms

Objective: The goal of this assignment is to apply and compare different reinforcement learning (RL) algorithms on the same game or environment. You will implement and evaluate the following algorithms:
1. Dynamic Programming
2. Monte Carlo Methods
3. Temporal Difference (TD) Learning
4. Q-Learning
5. SARSA
6. TD(λ) with Eligibility Traces

You will analyze their performance, strengths, and weaknesses in terms of convergence speed, computational complexity, and ability to handle delayed rewards.

Instructions

Step 1: Choose a Game or Environment
Select a simple game or environment that can be modeled as a Markov Decision Process (MDP). Please ensure that your chosen environment is **not the same as other students' choices** to encourage creativity and diversity in problem-solving. Examples include:
- Grid World: A grid-based environment where the agent navigates from a start state to a goal state while avoiding obstacles.
- Cart-Pole: A classic control problem where the agent must balance a pole on a cart.
- Frozen Lake: A slippery grid world where the agent must navigate from the start to the goal while avoiding holes.
- Custom Game: Design your own small-scale game with well-defined states, actions, rewards, and transitions.

Ensure the game satisfies the following criteria:
- Finite state and action spaces.
- Clear reward structure (e.g., positive rewards for goals, negative rewards for penalties).
- Deterministic or stochastic transitions.

Step 2: Implement the RL Algorithms
For the chosen game, implement the following algorithms:
1. Dynamic Programming:
   - Use Value Iteration or Policy Iteration to compute the optimal policy and value function.
   - Assume full knowledge of the environment's dynamics (transition probabilities and rewards).
2. Monte Carlo Methods:
   - Simulate episodes under a random or exploratory policy.
   - Estimate the value function by averaging returns over multiple episodes.
3. Temporal Difference (TD) Learning:
   - Use TD(0) to update the value function after each step.
   - Compare the results with Monte Carlo methods.
4. Q-Learning:
   - Implement Q-Learning to estimate the optimal action-value function.
   - Use an ε-greedy policy for exploration.
5. SARSA:
   - Implement SARSA to evaluate the current behavior policy.
   - Compare its performance with Q-Learning.

6. TD(λ) with Eligibility Traces:

- Extend TD Learning by incorporating eligibility traces.
- Experiment with different values of λ (e.g., $\lambda = 0.2, 0.5, 0.8$) to observe the impact on learning.

Step 3: Run Experiments
For each algorithm:
1. Set Parameters:
   - Discount factor (gamma): Typically 0.9.
   - Learning rate (alpha): Start with 0.1 and adjust as needed.
   - Exploration rate (epsilon): Start with 0.1 for ε-greedy policies.
   - Number of episodes or steps: Ensure sufficient iterations for convergence.
2. Record Metrics:
   - Convergence speed: How quickly does the algorithm find the optimal policy?
   - Computational cost: Measure runtime or memory usage.
   - Quality of the learned policy: Evaluate the policy's performance (e.g., average reward per episode).
3. Visualize Results:
   - Plot the value function or Q-values for each state or state-action pair.
   - Show the convergence of the algorithm over time (e.g., total reward per episode vs. number of episodes).

Step 4: Compare Results
Write a report comparing the performance of the algorithms. Address the following questions:
1. Convergence:
   - Which algorithm converges fastest? Why?
   - How does the choice of parameters (e.g., gamma, alpha, lambda) affect convergence?
2. Handling Delayed Rewards:
   - Which algorithm handles delayed rewards most effectively? Why?
   - How does TD(λ) compare to TD(0) and Monte Carlo methods in this regard?
3. Exploration vs. Exploitation:
   - How do Q-Learning and SARSA differ in balancing exploration and exploitation?
   - In what scenarios does SARSA outperform Q-Learning?
4. Computational Complexity:
   - Which algorithm is the most computationally efficient? Why?
   - How does the use of eligibility traces in TD(λ) impact memory and runtime?
5. Policy Quality:
   - Which algorithm produces the best policy (highest cumulative reward)?
   - Are there scenarios where suboptimal policies are acceptable (e.g., safety-critical environments)?

Step 5: Submit Your Work
1. Code:
   - Include implementations of all algorithms.
   - Provide clear comments and documentation.
2. Report:
   - Summarize the game/environment you chose.
   - Present experimental results (plots, tables, etc.).
   - Analyze and compare the performance of the algorithms.
   - Discuss the strengths and limitations of each algorithm.
3. Visualization:
   - Include visualizations of the value function, Q-values, and policy for each algorithm.
   - Highlight differences in the learned policies.