**Implementing a Reinforcement Learning Agent for a Gridworld Environment**

This project is to develop a **reinforcement learning (RL) agent** to navigate a **Gridworld environment** using a **model-free, off-policy, and value-based algorithm** called **Q-learning**. The agent must learn an optimal policy to maximize cumulative rewards while exploring the environment and avoiding obstacles. ***Appendix 1*** shows alternative Reinforcement Learning algorithms that can be further studied for this project.

**Objectives**

i. Develop a solid understanding of the **Markov Decision Process (MDP)** framework.

ii. Implement **Q-learning**, a fundamental RL algorithm.

iii. Train an RL agent to discover an **optimal policy** in a **discrete environment**.

iv. Evaluate and analyze the **performance** of the trained agent.

v. Visualize learning progression, Q-values, and the learned policy.

vi. Understand the impact of **exploration** vs. **exploitation** in reinforcement learning.

vii. Experiment with different **hyperparameter values** to analyze their effects on learning performance.

**Specifications**

**1. Environment Details**

A **5x5 Gridworld** where an agent must navigate from a **start position** to a **goal position**, avoiding obstacles and maximizing rewards.

- **Start state:** (0,0)
- **Goal state:** (4,4)
- **Obstacles:** Placed in predefined positions

**Rewards System:**

- **+10** for reaching the goal
- **-5** for hitting an obstacle
- **-1** for each step taken

**Notes:**

- The environment follows a deterministic transition model, meaning the agent's actions always lead to expected state transitions unless it hits an obstacle.

- The Gridworld is **fully observable**, so the agent always knows its current state.

**2. Reinforcement Learning Algorithm**

- **Q-learning** (an off-policy **Temporal Difference (TD) learning** algorithm)

- **Action Selection:** Uses an **ε-greedy strategy** to balance exploration and exploitation.

- **Hyperparameters:**
  - **Learning rate (α):** 0.1
  - **Discount factor (γ):** 0.9
  - **Exploration rate (ε):** Initially set to 1.0 and decays over time.
  - **Number of episodes:** Students should experiment with different episode counts (e.g., 1000, 5000) to observe learning convergence.

**Notes:**

- The **Bellman equation** will be used to update Q-values.

- The **exploration rate (ε)** should be decayed over time to allow the agent to shift from exploration to exploitation.

- Different **decay strategies** (e.g., linear decay, exponential decay) can be tested to evaluate their effect on learning.

**3. Implementation Requirements**

Implement the following:

- **Environment Representation**: Implement a **5x5 grid** with state transitions, rewards, and obstacles.

- **Q-learning Algorithm**: Implement Q-value updates using the **Bellman equation**.

- **Training Process**: Train the agent over multiple episodes to converge toward an optimal policy.

- **Evaluation Metrics**:
  - **Convergence analysis** (e.g., Q-values stabilization, cumulative reward trends)
  - **Agent performance** (number of steps to reach the goal, success rate)
  - **Exploration vs. exploitation balance**

- **Visualization of Results**:
  - **Heatmap** of learned Q-values
  - **Policy visualization** (arrows indicating optimal actions in each state)

**Notes:**

- Students are encouraged to modify and experiment with different **grid sizes, reward structures, and obstacle placements** to analyze the agent's adaptability.

- Implementing **additional evaluation metrics** (such as episode length variations) can provide deeper insights into learning efficiency.

**4. Expected Outputs**

- A **trained RL agent** capable of efficiently navigating the Gridworld. viii.

- A **Q-table** displaying learned Q-values for all state-action pairs.

- **Performance metrics**:
  - **Training progress graphs** (e.g., reward per episode, exploration-exploitation balance)
  - **Agent's success rate** and **average steps to reach the goal**.

- **Visualization tools**:
  - **Heatmap** of learned Q-values across the environment.
  - **Policy map** showing the best action in each state.
  - **Graphical representation of the training process** (optional but recommended for extra clarity).

**2. Submission Requirements**

- **Python Code**:
  - Well-documented and structured **Python implementation** of the Q-learning algorithm.
  - Proper use of functions, loops, and data structures for readability and efficiency.
  - Code should be modular, allowing for easy parameter adjustments and experimentation.

- **Report** including**:**
  - **Introduction**: Overview of reinforcement learning and Q-learning.
  - **Implementation details**:
    - Explanation of environment representation.
    - Algorithmic approach and parameter tuning.
    - Justification of chosen hyperparameters and their effects.
  - **Results & analysis**:
    - Learning curve discussion (with plots of reward trends over episodes).
    - Observations on Q-values, policy visualization, and training efficiency.
    - Comparison of different hyperparameter settings and their effects on learning.
  - **Challenges & Limitations**: Difficulties faced and potential improvements.
  - **Conclusion & Future Work**: Suggestions for enhancements or alternative methods.

---

**Notes:**

- Experiment with **different exploration strategies**, **hyperparameter tuning**, and **alternative visualization techniques** to enhance the understanding of RL concepts.

- Implementing additional **comparison studies** (e.g., testing different learning rates) can help deepen understanding of the impact of hyperparameters.

- For better presentation, provide a **short video demonstration** of the agent navigating the Gridworld.

- Compare Q-learning with other RL algorithms for additional insights.

**Appendix 1**

Implementing a Reinforcement Learning Agent for a Gridworld Environment using various Reinforcement Learning algorithms with respective sample objectives.

**Model-Free, Off-Policy, Value-Based**

1. Q-Learning- A model-free, off-policy reinforcement learning algorithm that updates Q-values using the maximum possible future reward, rather than the action actually taken. It enables agents to learn optimal policies in discrete environments like Gridworld.

**Model-Free RL Algorithms**

1. **SARSA (State-Action-Reward-State-Action)** – Similar to Q-learning but follows an on-policy approach, meaning it updates using the action actually taken instead of the best possible action.

Objectives:

- Understand the Markov Decision Process (MDP) framework.
- Implement SARSA, an on-policy RL algorithm.
- Train an agent to find an optimal policy while following an exploratory policy.
- Compare SARSA with Q-learning in terms of learning behavior and performance.

2. **Deep Q-Network (DQN)** – Uses a neural network to approximate the Q-table, making it suitable for larger or continuous state spaces.

Objectives:

- Understand function approximation in reinforcement learning.
- Implement a neural network-based Q-learning algorithm.
- Train an agent to learn optimal policies in environments with large or continuous state spaces.
- Utilize experience replay and target networks to stabilize training.

3. **Double Q-learning** – A variation of Q-learning that mitigates overestimation bias by using two Q-value estimates.

Objectives:

- Learn about overestimation bias in Q-learning.
- Implement Double Q-learning, which reduces bias by using two Q-value estimates.
- Train an agent to improve stability and performance compared to standard Q-learning.
- Analyze differences between Q-learning and Double Q-learning.

**Model-Based RL Algorithms**

4. **Dyna-Q** – Extends Q-learning by incorporating a model of the environment, allowing the agent to learn from simulated experiences.

Objectives:

- Understand the concept of model-based reinforcement learning.
- Implement Dyna-Q, which combines model-free learning with simulated experiences.
- Train an agent to improve sample efficiency by leveraging planning.
- Compare Dyna-Q with standard Q-learning in terms of convergence speed.

**Policy-Based Algorithms**

5. **Monte Carlo Policy Iteration –** Uses complete episodes to update value estimates and optimize policies.

Objectives:
- Learn about episodic learning in reinforcement learning.
- Implement Monte Carlo methods to estimate value functions and improve policies.
- Train an agent using complete episodes rather than step-by-step updates.
- Compare Monte Carlo learning with temporal difference (TD) methods.

6. **Policy Gradient Methods (REINFORCE)** – Directly optimize policies without requiring a value function.

Objectives
- Understand policy-based reinforcement learning.
- Implement REINFORCE, a Monte Carlo policy gradient method.
- Train an agent to directly optimize a policy without requiring a value function.
- Analyze the impact of learning rates and variance reduction techniques on training stability.

**Actor-Critic Methods**

7. **Advantage Actor-Critic (A2C)** – Combines value-based and policy-based methods to improve stability and performance.

Objectives
- Understand the combination of value-based and policy-based reinforcement learning.
- Implement the Advantage Actor-Critic (A2C) algorithm.
- Train an agent to learn an optimal policy using both value estimation and policy optimization.
- Compare A2C with other policy gradient and value-based approaches in terms of stability and efficiency.