# Contractor's Guide To RL

Kavya Gupta, Shravan S

January 22, 2024

# Contents

**1 Introduction** **3**

**2 Background** **3**

**3 Implementation** **3**
3.1 QAgent Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
3.2 Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
3.3 Training Loop . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
3.4 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
3.5 Testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

**4 Results** **4**
4.1 Fine-Tuning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
4.2 Rendering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

# 1 Introduction

Reinforcement learning (RL) is a branch of machine learning where an agent learns to make decisions by interacting with an environment. In this project, the goal is to solve the Mountain Car environment using Q-learning, a popular RL algorithm. The motivation is to understand how Q-learning can be applied to a continuous state space problem and analyze the learning performance of the agent.

# 2 Background

Reinforcement learning involves an agent learning to make decisions by receiving feedback in the form of rewards or penalties. Q-learning is a model-free RL algorithm that learns to associate actions with states to maximize cumulative rewards. The Gymnasium library provides a convenient interface for creating RL environments. The Mountain Car environment presents a challenge where a car must reach the flag at the top of a hill, requiring the agent to learn a strategy for efficient movement.

# 3 Implementation

## 3.1 QAgent Class

The `QAgent` class encapsulates the functionality of the RL agent. It initializes the Gymnasium environment, defines the observation space size, action space size, and hyperparameters such as learning rate ($\alpha$) and discount factor ($\gamma$). The Q-table is initialized with random values to represent state-action pairs.

## 3.2 Methods

- `get_state_index`: This method discretizes the continuous state space into indices for the Q-table using the provided formula.
- `update`: The `update` method implements the Q-learning update rule. It calculates the new Q-value based on the reward, the maximum Q-value for the next state, and the current Q-value.
- `get_action`: The `get_action` method implements an epsilon-greedy strategy. With probability $\epsilon$, it selects a random action for exploration; otherwise, it exploits the action with the highest Q-value.
- `env_step`: This method takes a step in the environment, updates the Q-table, and moves to the next state.

### 3.3 Training Loop

The training loop iterates over a specified number of episodes. Within each episode, the agent interacts with the environment using the epsilon-greedy strategy. The Q-table is updated after each step based on the reward and the Q-learning update rule. The epsilon value is decayed linearly over episodes.

### 3.4 Evaluation

The `agent_eval` method is used to visualize the performance of the trained agent. This allows a qualitative assessment of the learned policy.

### 3.5 Testing

The `test_agent` method is used to evaluate the agent's performance over a specified number of episodes.

## 4 Results

### 4.1 Fine-Tuning

Hyperparameter fine-tuning experiments were conducted to optimize parameters such as learning rate ($\alpha$), discount factor ($\gamma$), and the discretization sizes. The impact of these parameters on learning performance was analyzed.

### 4.2 Rendering

The rendering frequency during training was adjusted to balance visualization and computational efficiency. Rendering allows monitoring the agent's behavior in real-time.