WiDS 2023 Report: Contractor's Guide To RL

Kavya Gupta, Shravan S

January 22, 2024

Contents

1 Mountain Car Game

1.1 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.

1.2 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.

1.3 Implementation

1.3.1 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 (α) and discount factor (γ). The Q-table is initialized with random values to represent state-action pairs.

1.3.2 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 ϵ, 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.

1.3.3 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.

1.3.4 Evaluation

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

1.3.5 Testing

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

1.4 Results

1.4.1 Fine-Tuning

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

1.4.2 Rendering

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

2 N-Armed Bandits Problem

In addition to the Mountain Car environment, we explored the n-armed bandits problem—a classic reinforcement learning scenario where an agent must choose among several actions, each associated with an unknown reward distribution. The goal is to maximize the cumulative reward over time.

2.1 Agent Classes

We implemented several agents to tackle the n-armed bandits problem, each employing a different strategy:

2.1.1 GreedyAgent

The `GreedyAgent` always chooses the action with the highest estimated value based on historical rewards. It maintains a Q-value ($Q(a)$) for each action.

2.1.2 epsGreedyAgent

The `epsGreedyAgent` balances exploration and exploitation by choosing a random action with probability ϵ and otherwise selecting the action with the highest Q-value.

2.1.3 UCBAAgent

The `UCBAAgent` employs the Upper Confidence Bound (UCB) strategy, which prioritizes actions based on the estimated mean reward ($\hat{\mu}(a)$) and an exploration term ($U(a)$):

$$U(a) = \sqrt{\frac{2\ln(t)}{N(a)}}$$

where $N(a)$ is the number of times action a has been selected, and t is the total number of iterations.

2.1.4 GradientBanditAgent

The `GradientBanditAgent` uses a softmax function to select actions based on their estimated preferences ($H(a)$):

$$\pi(a) = \frac{e^{H(a)}}{\sum_{i=1}^{k} e^{H(i)}}$$

It updates action preferences based on rewards received using the update rule:

$$H(a) \leftarrow H(a) + \alpha(R - \bar{R})(1 - \pi(a))$$

where R is the received reward, $\bar{R}$ is the average reward, and α is the learning rate.

2.1.5 ThompsonSamplerAgent

The `ThompsonSamplerAgent` employs Bayesian methods, specifically the Thompson sampling strategy, to update its belief about the reward distribution for each action. It uses Beta-distributed priors to model success and failure counts ($\alpha(a), \beta(a)$):

$$P(R(a)|\alpha(a), \beta(a)) = \text{Beta}(\alpha(a) + \text{successes}(a), \beta(a) + \text{failures}(a))$$

The action is selected based on the sampled values from the posterior distribution.

2.2 Performance Evaluation

To evaluate the performance of the different agents, we conducted experiments over a specified number of iterations. The cumulative rewards obtained by each agent were recorded and analyzed.

2.3 Performance Graph

The performance graph (see Figure 1) illustrates the cumulative rewards obtained by each agent over time.

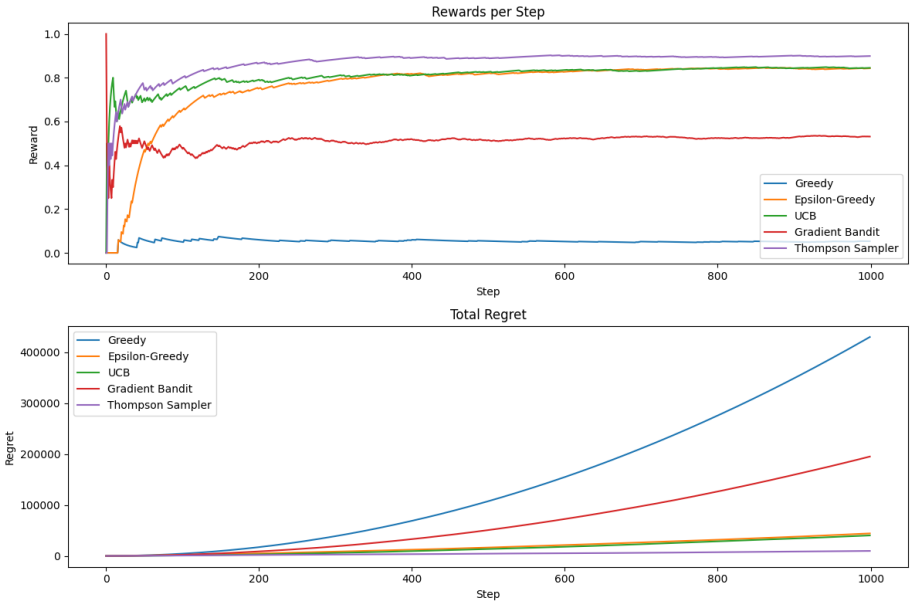

Figure 1: Cumulative rewards obtained by n-armed bandits agents over iterations.

The graph provides insights into how well each agent learned to exploit the actions with higher rewards and adapt its strategy over time.

2.4 Comparison with Mountain Car Results

In contrast to the continuous state space of the Mountain Car environment, the n-armed bandits problem focuses on discrete actions and rewards. Comparing the results and strategies employed by agents in both scenarios can offer valuable insights into the adaptability and generalization capabilities of reinforcement learning algorithms.