Solving the Mountain Car Problem with Q-Learning

Theophilus Dwamena Frimpong

Postgraduate Diploma in Quantitative Life Sciences Student
The Abdus Salam International Center for Theoretical Physics (ICTP)

June 30, 2025

Overview

1. What is the Mountain Car problem?
2. Reinforcement Learning formulation
3. Q-Learning approach
4. Results and interpretation
5. Limitations and alternatives

What is the Mountain Car Problem?

Problem Description

Imagine a car positioned in a one-dimensional valley between two hills. The car's engine is not powerful enough to drive directly up the steeper right hill where the goal flag is located.

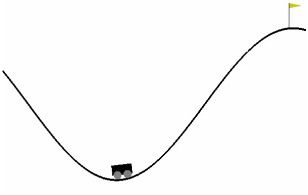

Figure: *Illustration of the Mountain Car environment.*

- **Goal:** The car must learn a policy to reach the flag.

To frame the Mountain Car Problem, we define its key components:

- **State space ($\mathcal{S}$):**
  - Position $x \in [-1.2, 0.6]$
  - Velocity $\dot{x} \in [-0.07, 0.07]$

  Therefore, a state $s \in \mathcal{S}$ can be represented as a tuple $(x, \dot{x})$.

Reinforcement Learning Formulation

RL Framework (1/2)

To frame the Mountain Car Problem, we define its key components:

- **State space ($\mathcal{S}$):**
  - Position $x \in [-1.2, 0.6]$
  - Velocity $\dot{x} \in [-0.07, 0.07]$

  Therefore, a state $s \in \mathcal{S}$ can be represented as a tuple $(x, \dot{x})$.

- **Action space ($\mathcal{A}$):**
  - -1 (Push left)
  - 0 (No push)
  - $+1$ (Push right)

  So $\mathcal{A} = \{-1, 0, +1\}$. (denoted as $\{0, 1, 2\}$ in python)

To frame the Mountain Car Problem, we define its key components:

- **State space ($\mathcal{S}$):**
  - Position $x \in [-1.2, 0.6]$
  - Velocity $\dot{x} \in [-0.07, 0.07]$

  Therefore, a state $s \in \mathcal{S}$ can be represented as a tuple $(x, \dot{x})$.

- **Action space ($\mathcal{A}$):**
  - -1 (Push left)
  - 0 (No push)
  - $+1$ (Push right)

  So $\mathcal{A} = \{-1, 0, +1\}$. (denoted as $\{0, 1, 2\}$ in python)

- **Model [$p(s^{'}|s, a)$]:**

$$\dot{x}_{t+1} = \dot{x}_t + 0.0015 a_t - 0.0025 \cos(3x_t); \qquad x_{t+1} = x_t + \dot{x}_{t+1},$$

where a_t is the current action and $\mathrm{power} = 0.0015$.

- **Rewards:**
  - Reaching the flag ($x \geq 0.45$) terminates the episode ($+100$).
  - Failure to reach the flag (goal) after each time step results in a -1 reward. After 300 failed steps, the episode truncates.
  - A negative reward of $-0.1 \cdot action^2$ is received at each timestep to penalise for taking actions.

$$R_t = R^{(t)} - 0.1 a_t^2 \ ; \qquad R^{(t)} = \begin{cases} +100 & \text{if goal is reached,} \\ -1 & \text{if goal is not reached,} \end{cases}$$

where R_t is the total reward for each time step.

- **Rewards:**
  - Reaching the flag ($x \geq 0.45$) terminates the episode ($+100$).
  - Failure to reach the flag (goal) after each time step results in a -1 reward. After 300 failed steps, the episode truncates.
  - A negative reward of $-0.1 \cdot action^2$ is received at each timestep to penalise for taking actions.

$$R_t = R^{(t)} - 0.1a_t^2\ ; \qquad R^{(t)} = \begin{cases} +100 & \text{if goal is reached,} \\ -1 & \text{if goal is not reached,} \end{cases}$$

where R_t is the total reward for each time step.

- **Agent:** A control system with sensors measuring x and $\dot{x}$.

- **Rewards:**
  - Reaching the flag ($x \geq 0.45$) terminates the episode ($+100$).
  - Failure to reach the flag (goal) after each time step results in a -1 reward. After 300 failed steps, the episode truncates.
  - A negative reward of $-0.1 \cdot \mathit{action}^2$ is received at each timestep to penalise for taking actions.

$$R_t = R^{(t)} - 0.1 a_t^2 \; ; \qquad R^{(t)} = \begin{cases} +100 & \text{if goal is reached,} \\ -1 & \text{if goal is not reached,} \end{cases}$$

  where R_t is the total reward for each time step.

- **Agent:** A control system with sensors measuring x and $\dot{x}$.

- **Observations:** At each time step, the agent observes the state and the reward. (initial state is $S(x_0 \in [-0.6, 0.4], \ \dot{x}_0)$)

Q-Learning is a model-free, off-policy reinforcement learning algorithm.

- It learns the **action-value function** $Q(s, a)$, which estimates expected future rewards from taking action a in state s, and then acting optimally.
- The agent improves its policy by updating $Q(s, a)$ through interaction with the environment, without needing a model of transitions or rewards.
- The optimal policy is derived by choosing the action that maximizes $Q(s, a)$ (or some different strategy):

$$\pi^*(s) = \begin{cases} \arg\max_a Q(s, a) & \textbf{w.p } 1-\epsilon \text{ (Exploitation)} \\ \text{random } a & \textbf{w.p } \epsilon \text{ (Exploration)} \end{cases}$$

Pseudocode:

- Initialize $Q(s, a)$ arbitrarily
- For each episode:
  - Initialize state s
  - Repeat (for each step in the episode):
    - Choose a using ϵ-greedy policy (with decay over time)
    - Take action a, observe reward r and next state s'
    - TD Update **(Bellman Optimality)**:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

    - $s \leftarrow s'$;

  until s is terminal

- **Addressing Continuous State Space:**
  The Mountain Car's continuous state space renders tabular Dynamic Programming methods impractical, but by discretizing the environment, Q-Learning offers a computationally feasible and effective solution.

Q-Learning Approach

Why Q-Learning?

- **Addressing Continuous State Space:**
  The Mountain Car's continuous state space renders tabular Dynamic Programming methods impractical, but by discretizing the environment, Q-Learning offers a computationally feasible and effective solution.

- **Model-Free Learning:**
  Q-Learning offers a major advantage in complex or unknown environments by learning optimal behavior through direct interaction, without needing a predefined model of the system's dynamics $p(s'|s,a)$ or reward structure.

- **Addressing Continuous State Space:**
  The Mountain Car's continuous state space renders tabular Dynamic Programming methods impractical, but by discretizing the environment, Q-Learning offers a computationally feasible and effective solution.

- **Model-Free Learning:**
  Q-Learning offers a major advantage in complex or unknown environments by learning optimal behavior through direct interaction, without needing a predefined model of the system's dynamics $p(s'|s, a)$ or reward structure.

- **Off-Policy Control:**
  Q-Learning's off-policy nature allows it to learn the optimal policy from experiences gathered through a different behavior policy, making it highly adaptable and efficient in utilizing exploratory data.

- **State Space Discretization:**
  - Transforms continuous state variables into a finite, discrete grid.
  - Each continuous dimension is divided into 25 'bins'.
  - A continuous state $(x, \dot{x})$ maps to a unique 2D grid index.
  - **Total Discrete States:** $25 \times 25 = 625$.

Q-Learning Approach

Exploration and Discretization Strategy

- **State Space Discretization:**
  - Transforms continuous state variables into a finite, discrete grid.
  - Each continuous dimension is divided into 25 'bins'.
  - A continuous state $(x, \dot{x})$ maps to a unique 2D grid index.
  - **Total Discrete States:** $25 \times 25 = 625$.
- **Q-Table Structure:**
  - With 3 possible actions for each discrete state, the Q-table has a shape of 625×3.

Q-Learning Approach

Exploration and Discretization Strategy

- **State Space Discretization:**
  - Transforms continuous state variables into a finite, discrete grid.
  - Each continuous dimension is divided into 25 'bins'.
  - A continuous state $(x, \dot{x})$ maps to a unique 2D grid index.
  - **Total Discrete States:** $25 \times 25 = 625$.
- **Q-Table Structure:**
  - With 3 possible actions for each discrete state, the Q-table has a shape of 625×3.
- **Epsilon-Greedy Exploration:**
  - Balances exploration (trying new actions) and exploitation (using known optimal actions).
  - Initial $\epsilon = 1.0$ (pure exploration) decays over time, ensuring convergence to an optimal policy.

Results and Interpretation
Training

After sufficient training, the agent reliably learns a policy to reach the goal.

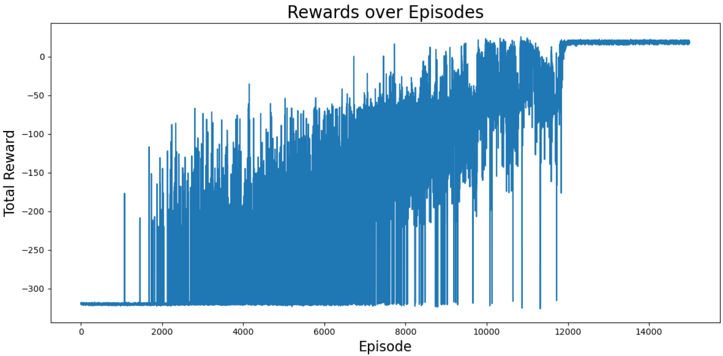

Figure: Training progression of the agent over episodes.

Convergence after ∼12,000 episodes.

Final (learned) updated Q-values, thus $Q^*(s, a)$:

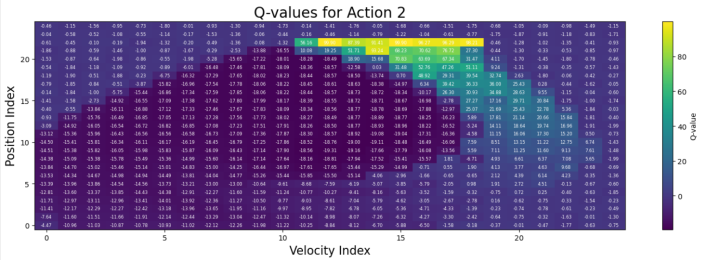

Figure: Learned action-value function for action 2 (right push).

These plots reveal regions where the agent deems it optimal to remain idle or reverse direction.

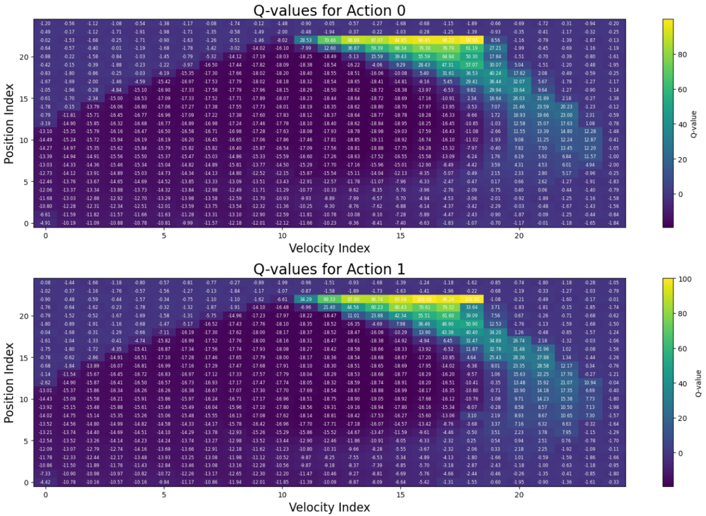

Figure: Learned action-value function for actions 0 (left push) and 1 (no push).

Results and Interpretation

Learned Policy

- Strategy ($x_0 \in [-0.6, -0.4]$) :
  - The car alternates between pushing left and right to climb the hills and build momentum.
  - With each swing, it gains more speed until it can reach the flag.

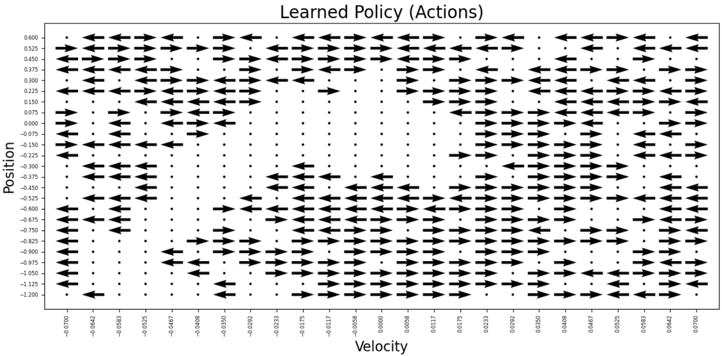

Figure: Learned policy map.

Consistent with expectations.

To quantify the learning process, we track the rewards the agent receives over thousands of training episodes.

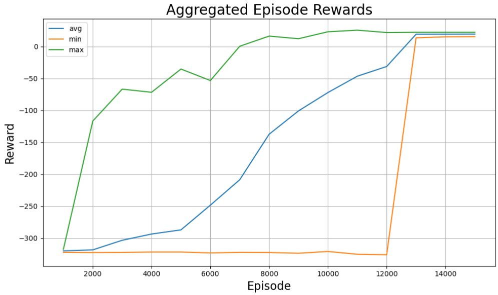

Figure: Raw aggregated episode rewards.

```
[20]:
agent.agent_eval()

Evaluation started (no render for custom env)
Evaluation reward: 20.50
```

Limitations and Alternatives

- **Limitations of the Tabular Q-Learning Approach:**
  - Performance is heavily reliant on the chosen grid resolution.
  - Tabular Q-Learning does not scale well to problems with very high-dimensional state spaces, as the Q-table size grows exponentially.
  - Relies on exploration strategy like ϵ-greedy, which is effective but inefficient due to its inherent randomness.

Limitations and Alternatives

- **Limitations of the Tabular Q-Learning Approach:**
  - Performance is heavily reliant on the chosen grid resolution.
  - Tabular Q-Learning does not scale well to problems with very high-dimensional state spaces, as the Q-table size grows exponentially.
  - Relies on exploration strategy like ϵ-greedy, which is effective but inefficient due to its inherent randomness.
- **Alternative Approaches for Reinforcement Learning:**
  - SARSA (State-Action-Reward-State-Action):
  - Deep Q-learning (function approximations).
  - Actor-Critic (for the continuous actions case).

- **Summary of Achievement:**
  - Used Q-Learning with discretization to effectively tackle the Mountain Car problem in a continuous state space.
  - The agent learned an optimal, intuitive policy that leverages momentum to efficiently reach the goal.
  - This project showcased the power of Q-Learning as a model-free and off-policy reinforcement learning algorithm.

References

A. W. Moore, “Efficient Memory-based Learning for Robot Control,” University of Cambridge Technical Report, 1990. Available at: `https://gymnasium.farama.org/environments/classic_control/mountain_car_continuous/`

A. M. Andrew, “REINFORCEMENT LEARNING: AN INTRODUCTION by Richard S. Sutton and Andrew G. Barto,” *Robotica*, vol. 17, no. 2, pp. 229–235, 1999. MIT Press, Cambridge, MA. ISBN 0-262-19398-1.

Thank You!

Prepared by Theophilus Dwamena Frimpong

`https://github.com/Theophilus-Dwamena/TheMountainCarProblemQLearning`

Questions?