WiDS Kalman Filtered Trend Trader: Assignment 3

Aarav Malde and Krishang Krishna

25th December, 2025

Task 1: Environment Setup

Set up the Python environment required for reinforcement learning using `Gymnasium`. Ensure that the environment runs correctly before implementing any learning algorithm.

We strongly recommend using a virtual environment.

Step 1: Create and Activate a Virtual Environment

Windows (PowerShell or Command Prompt):

```
python -m venv rl-env
rl-env\Scripts\activate
```

macOS / Linux (Terminal):

```
python3 -m venv rl-env
source rl-env/bin/activate
```

After activation, your terminal prompt should indicate that the virtual environment is active.

Step 2: Install Required Packages

Install `Gymnasium` and its classic control environments:

```
pip install gymnasium
```

If this does not work properly, immediately contact Me or Aarav.

Step 3: Verify the FrozenLake Environment

Run the following minimal script to confirm that the environment initializes correctly and accepts actions:

```
import gymnasium as gym

env = gym.make("FrozenLake-v1")
state, info = env.reset()

print("Initial state:", state)
print("State space:", env.observation_space)
print("Action space:", env.action_space)

env.close()
```

If this runs without errors, your environment is correctly set up. You should see the Frozen Lake pop up for one game.

Task 2: Tabular Q-Learning on FrozenLake

In this task, you will implement **tabular Q-learning** for the FrozenLake environment. You are provided with a complete implementation skeleton. Your job is to fill in the missing parameters and observe how learning behavior changes.

You are **not** expected to modify the environment or the algorithm structure. The objective is to understand how parameter choices affect learning.

Q-Learning Skeleton

Open the `Frozen_Lake_Skeleton.py` file provided.

What You Should Observe

After filling in the parameters and running the code:

- The success rate should start near zero.
- Learning should be slow and noisy.
- Final performance should not reach perfect success due to stochasticity.

You are expected to experiment with different parameter values and analyze how they affect convergence and stability.

Submission

Submit only the Parameter values, no need to send the whole code.

Task 3: MountainCar and the Need for Deep Q-Learning

In this task, you will work with the **MountainCar** environment. Unlike FrozenLake, this environment involves *continuous state variables* and long-term planning under delayed rewards.

The MountainCar Environment

The MountainCar environment consists of a car positioned in a valley between two hills. The engine of the car is not powerful enough to drive directly to the goal at the top of the right hill. To succeed, the agent must learn to move away from the goal initially in order to build sufficient momentum.

Key properties of the environment:

- The state consists of **continuous** variables (position and velocity).
- The action space is discrete (push left, do nothing, push right).
- The reward is -1 at every time step until the goal is reached.
- The optimal strategy requires *temporarily worsening* the state in order to achieve long-term success.

Attempting Q-Learning

I encourage you to try the same Q-Learning Loop used in FrozenLake here.

However, because the state space is continuous, **tabular Q-learning is not directly applicable**. The simplest workaround is to discretize the state space into bins and treat the problem as approximately discrete. If you experiemt with this approach, you should observe that:

- Learning is highly sensitive to the choice of discretization and increasing resolution rapidly increases memory and computation costs.
- Performance remains unstable and often poor.

This failure is expected and is a consequence of poor generalization in tabular methods when applied to continuous spaces.

Motivation for Deep Q-Networks (DQN)

To address these limitations, we replace the tabular representation of the action-value function with a **neural network**. Instead of storing a value for every state–action pair, the network learns a function

$$Q(s, a) \approx Q_\theta(s, a)$$

that generalizes across similar states.

Deep Q-Networks (DQN) introduce several key ideas:

- **Function approximation** using neural networks to handle continuous state spaces.
- **Experience replay**, which stabilizes learning by breaking temporal correlations.
- **Target networks**, which reduce feedback loops during training.

These mechanisms allow the agent to learn from delayed rewards and propagate value information backward over long time horizons, which is essential for solving the MountainCar task.

Task Description

Your objective is to modify your learning loop to implement **Deep Q-Learning** for the MountainCar environment. You are not expected to achieve perfect performance, but you should observe a clear improvement over time and qualitatively sensible behavior.

Hints

- Exploration must persist for a long time in this environment; decaying exploration too quickly often leads to failure.
- Delayed rewards require careful tuning of the discount factor and learning rate for stable training.

You are expected to analyze the learning behavior, discuss failure modes, and explain how function approximation improves performance relative to tabular methods.

Submission

Submit your entire code this time.

May the Profit be with you!