Com S 4740/5740 Introduction to Machine Learning Homework 5

Laith Al Sairafi

December 17th, 2025

1 Introduction

For this assignment, an agent and environment were implemented for reinforcement learning purposes. The environment simulated a 1D rocket landing with a 2D state vector that contains the current velocity and altitude, and scalar action, which is the thrust force applied. For the agent, DQN was chosen due to its suitability for continuous input spaces, which is applicable to this environment and many others. The external API of the provided skeleton to the environment and agent classes didn't change, meaning that the method's signatures and returns are stills the same.

2 Environment Implementation

For the environment, the provided template was used as a wrapper class for the `RocketLandingEnv` class, which implements several methods used to initialize and reset a rocket landing, retrieve the current state, and step. A piecewise reward function was implemented to facilitate learning the desired behavior.

The `RocketLandingEnv` class constructor inputs are:

- **max_thrust**: the max thrust in N that the rocket can generate (default = 20).
- **drag_constant**: the drag constant, which is used in the acceleration calculation (default = 1).
- **max_velocity**: the maximum rocket initial random velocity (default = 20).
- **max_altitude**: the maximum rocket initial random altitude (default = 50).
- **dt**: stimestep for integration (default = 0.1).
- **m**: the mass of the rocket (default = 10).
- **g**: the gravity acceleration (default = 10).

The environment's default parameters are used in the submitted code.

The state of the rocket is described in the 2D vector, which contains the velocity and the altitude of the rocket (i.e., $s \in \mathbb{R}^2$) and a scalar action (i.e., $a \in \mathbb{R}$) which describes the thrust force applied to slow down the rocket. As required by the assignment, the action received from the agent (i.e., $a \in [0, 1]$) is de-normalized to the thrust range ($[0, \text{max\_thrust}]$), and the returned state is normalized (i.e., $s \in [0, 1]^2$).

The reward function is implemented as follow (a = altitude, v = velocity, and t = thrust):

$$r(s,a) = \begin{cases} -(0.002a^2 + 0.001v^2 + 0.04t^2), & \text{if} \quad t > 5 \\ -(0.002a^2 + 0.002v^2 + 0.005t^2), & \text{if} \quad 1 < t \leq 5 \\ 100(1-a)^2 - (0.004v^2 + 0.001t^2), & \text{if} \quad 0 \geq t \leq 1 \\ -100, & \text{otherwise} \end{cases} \tag{1}$$

The goal was to punish high altitude, and while the altitude is greater than 1 unit of distance, the punishment for high speed increases as the rocket approaches 0 units of distance in altitude, while also decreasing the punishment for using the thrust. When the rocket is in the desired altitude range (0 to 1 unit of distance), the agent starts get get positive reward, and the reward increases the closer the rocket is to 0 altitude, velocity, and thrust. If the rocket crashes (i.e., negative altitude), the reward is -100.

3 Agent Implementation

In the agent's implementation, vanilla DQN is used for the advantages that having a continuous state space offers, compared to traditional tabular Q-Learning. The agent used a randomly samples replay buffer that stores up to 10000 previous actions. The target network is updated every 200 steps (after each episode). For the policy network training, Adam, a variant of stochastic gradient descent (SGD), is used with a batch size of 128 samples. For the epsilon decay, it starts at 1 and ends at 0.1 with a decay rate of 0.995. The state dimension is 2, as required by the assignment, and the action dimension is 21 (fixed, not dynamic, unfortunately). 128 neurons are used in each hidden layer of the policy and target networks. Lastly, the learning rate is 0.001.

Again, same as the environment, the provided agent class template was used as a wrapper class, and another class named `DQNAgent` was created that contains the implementation of the DQN.

The `DQNAgent` class constructor inputs are:

- **state_dim**: state dimension size. (default = 2).
- **action_dim**: action dimension size. (default = 1).
- **hidden_dim**: Number of neurons in each hidden layer of the neural network. (default = 128).
- **learning_rate**: Step size used by the optimizer (default = 0.001).
- **discount_factor**: Determines the importance of future rewards (default = 0.99).
- **epsilon_start**: Initial exploration probability (default = 1).
- **epsilon_end**: end exploration probability (default = 0.1).
- **epsilon_decay**: Multiplicative decay factor (default = 0.995).
- **batch_size**: Number of experience samples randomly drawn (default = 128).
- **buffer_size**: Maximum number of transitions stored in the replay buffer (default = 10000).
- **target_update_freq**: Number of training steps between updates of the target network (default = 200).

The agent's default parameters are used in the submitted code.

It is also worth noting that updating the policy network doesn't start until three times the batch size of samples has been appended to the replay buffer, allowing for some randomness.