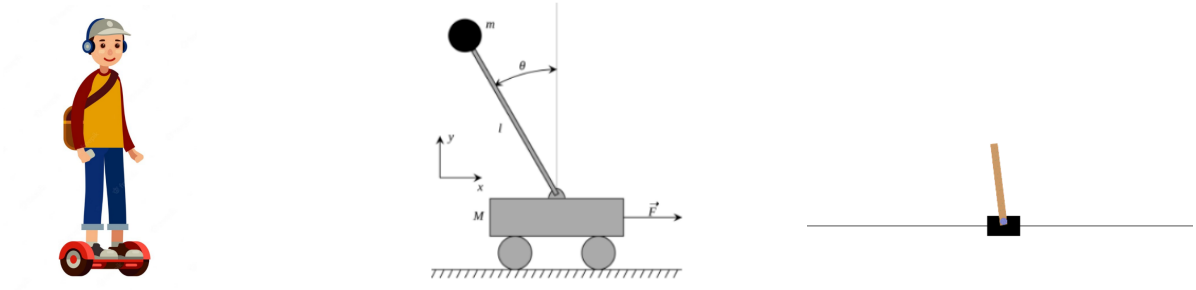
CSC3180: Fundamentals of Artificial Intelligence, Spring 2023
Project Report, Group 10
April 29, 2023

Adaptive Dynamic Stabilization of a Self-Balancing Scooter Using Epsilon-Greedy Q-learning

Hongyi Yang, Muhan Lin, Xuanyang Xu, Zixing Jiang

1 Introduction

The self-balancing scooter (Figure 1a) is a common urban commuter tool. One of the most critical features in self-balancing scooter is dynamic stabilization, which allows the scooter to maintain its balance while in motion. Without this feature, it would be very difficult for riders to stay upright and control the scooter, and even cause safety accidents in severe cases. In order to achieve dynamic stabilization, the traditional automatic control methods, including Proportional-Integral-Derivative (PID) control and Model Predictive Control (MPC), model the self-balancing scooter as a cart-pole system (Figure 1b) and designs the controller by deriving its system dynamics. However, the dynamics-based control is sensitive to the rider's weight and center of mass (CoM). For riders of different weights and CoMs, the performance of stabilization varies. In this project, we aim to utilize the learning ability of artificial intelligence to design a dynamic stabilizer that adapts to different rider weights and CoMs.



(a) Man on a self-balancing scooter (b) Cart-pole system dynamics (c) OpenAI Cart Pole Environment

Figure 1: Illustrations of (a) the self-balancing scooter, (b) the cart-pole system dynamics, and (c) the OpenAI Cart Pole environment.

2 Problem Statement

In this project, we model the scooter as the cart-pole system (Figure 1b). The wheel of scooter is the cart the rider-scooter mass is the pole. Then the adaptive stabilization problem can be formulated as follows: control the inclination (θ) of a pole with varying mass and CoM within a certain range by controlling the movement (x) of the cart (leftward or rightward). We intend to apply reinforcement learning (RL) to solve this problem. Namely, given the observation of the environment $(\theta, \dot{\theta}, x, \dot{x})$, we want our agent (cart) to learn a policy (move leftward or move rightward) to stable the varying pole through trail and reward. We plan to implement our RL algorithm based on the OpenAI Gym environment for cart-pole (Figure 1c).

3 Epsilon-Greedy Q-Learning

In this project, we built our dynamic stabilization controller with Epsilon-Greedy Q-Learning, which is a well-developed reinforcement learning algorithm that balances exploration and exploitation in sequential decision-making problems. In this section, we will introduce the Epsilon-Greedy Q-Learning algorithm in detail and explain how we link this algorithm to the dynamic balancing problem we aim to solve.

States \ Actions	$\boldsymbol{a}_0$	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	...
$\boldsymbol{s}_0$	$Q(\boldsymbol{s}_0, \boldsymbol{a}_0)$	$Q(\boldsymbol{s}_0, \boldsymbol{a}_1)$	$Q(\boldsymbol{s}_0, \boldsymbol{a}_2)$	...
$\boldsymbol{s}_1$	$Q(\boldsymbol{s}_1, \boldsymbol{a}_0)$	$Q(\boldsymbol{s}_1, \boldsymbol{a}_1)$	$Q(\boldsymbol{s}_1, \boldsymbol{a}_2)$	...
$\boldsymbol{s}_2$	$Q(\boldsymbol{s}_2, \boldsymbol{a}_0)$	$Q(\boldsymbol{s}_2, \boldsymbol{a}_1)$	$Q(\boldsymbol{s}_2, \boldsymbol{a}_2)$	...
$\boldsymbol{s}_3$	$Q(\boldsymbol{s}_3, \boldsymbol{a}_0)$	$Q(\boldsymbol{s}_3, \boldsymbol{a}_1)$	$Q(\boldsymbol{s}_3, \boldsymbol{a}_2)$	...
...	...	...	...	...

Table 1: An example of Q-Table. $Q(\boldsymbol{s}_i, \boldsymbol{a}_j)$ denotes the Q-Value of a state-action pair $(\boldsymbol{s}_i, \boldsymbol{a}_j)$. The higher $Q(\boldsymbol{s}_i, \boldsymbol{a}_j)$ is, the more worthy to take action $\boldsymbol{a}_j$ at state $\boldsymbol{s}_i$.

3.1 Q-Learning

Q-Learning is a classic value-based reinforcement learning algorithm that allow an agent to learn from its own actions and rewards in an environment. Unlike some other model-based control and reinforcement learning methods, it does not require a model of the environment. This feature helps to solve the traditional model-based control method's dependence on the rider's center of gravity mentioned in previous sections.

The goal of Q-Learning is to find an optimal policy that maximizes the expected value of the total reward over any sequence of actions. The main idea of Q-Learning is to use a table, called Q-Table, that stores the estimated value $Q(\boldsymbol{s}, \boldsymbol{a})$ (Q-Value) of each action $\boldsymbol{a}$ in each state $\boldsymbol{s}$, as shown in Table 1. The Q-Value, which reflects the value of taking a certain action in a certain state, is defined as the expectation of overall received rewards after taking action $\boldsymbol{a}$ at state $\boldsymbol{s}$, as shown in (1).

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \mid \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\right) \tag{1}$$

where $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the reward received from the environment for state-action pair $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and γ is the discount factor.

In Q-Learning, in order to obtain the most accurate estimate of the Q-Value, the Bellman Equation is applied to update $Q(\boldsymbol{s}, \boldsymbol{a})$ in every iteration that state-action pair $(\boldsymbol{s}, \boldsymbol{a})$ is chosen, as shown in (2)

$$\text{New } Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow (1-\lambda)\text{Current } Q(\boldsymbol{s}, \boldsymbol{a}) + \lambda\left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a'}} Q(\boldsymbol{s'}, \boldsymbol{a'})\right] \tag{2}$$

where $r(\boldsymbol{s}, \boldsymbol{a})$ is the reward received in this iteration, $\boldsymbol{s'}$ is the transient state after taking $\boldsymbol{a}$ at $\boldsymbol{s}$, $\boldsymbol{a'}$ is the legal action at $\boldsymbol{s'}$, γ is the discount factor, and λ is the learning rate.

In the Q-Learning process, the agent updates the Q-Table as it explores the environment and receives feedback. The agent chooses the action $\boldsymbol{a}^*(\boldsymbol{s})$ that has the highest value in the current state $\boldsymbol{s}$, according to the Q-Table, as shown in (3).

$$\boldsymbol{a}^*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{3}$$

This way, the agent learns to act optimally over time.

In our scenario, the state vector $\boldsymbol{s}$ of the cart-pole system is continuous. In order to prevent infinitely many state-action pairs, we discretized the system's state space by intervals. The length of each interval is fine-tuned to ensure the effectiveness of learning.

3.2 Epsilon-Greedy Action Selection

In order to learn an optimal Q-table, we want our algorithm to explore all possible state-action pairs. Thus, in addition to the action selection criterion (3), an epsilon-greedy policy is applied. Given an epsilon-greedy factor $0 \leq \varepsilon \leq 1$, in each iteration, there is a probability of ε that the agent takes random actions to explore new possible state-action pairs, and there is a probability of $1 - \varepsilon$ that the agent exploits investigated state-action pairs via the Q-Table. The overall logic flow of the Epsilon-Greedy Q-Learning is shown in algorithm 1.

Algorithm 1 Epsilon-Greedy Q-Learning

Require: observation of state $\boldsymbol{s}$, epsilon-greedy factor $\varepsilon \in [0, 1]$
  while learning not terminate **do**
    Generate a random number $i \in [0, 1]$
    if $i \leq \varepsilon$ **then**
      Take random action $\boldsymbol{a}$
    else
      Select action $\boldsymbol{a}$ according to the Q-Table ▷ Equation (3)
    end if
    Update Q-Table entry for $(\boldsymbol{s}, \boldsymbol{a})$ according to the received reward $r(\boldsymbol{s}, \boldsymbol{a})$ ▷ Equation (2)
  end while