Reinforcement Learning to Simulate Infinitely Repeated Games

CS 136: Economics and Computation

Iñaki Arango & Kelsey Chen

What is Reinforcement Learning?

“RL is the science of using experiences to make optimal decisions”

Steps:

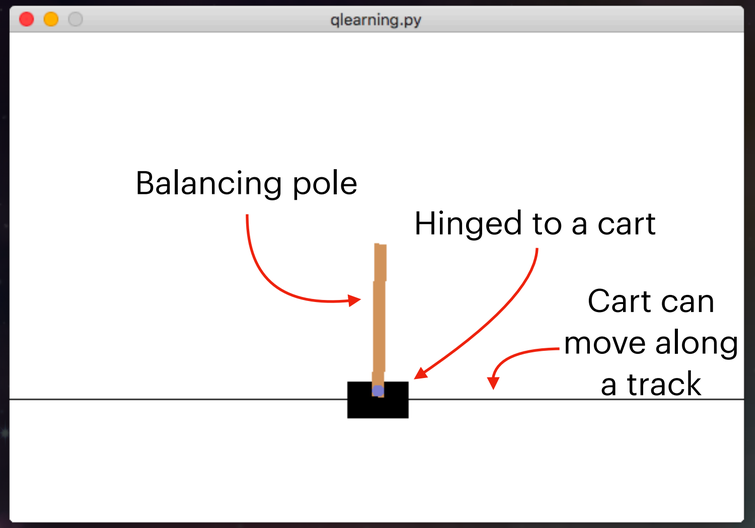
- Observing the environment
- Making decisions on how to act using some strategy
- Executing the action
- Receiving a reward or penalty based on the action
- Learning from experiences and refining the strategy
- Repeating this iterative process until an optimal strategy is found

Reinforcement Learning Cycle

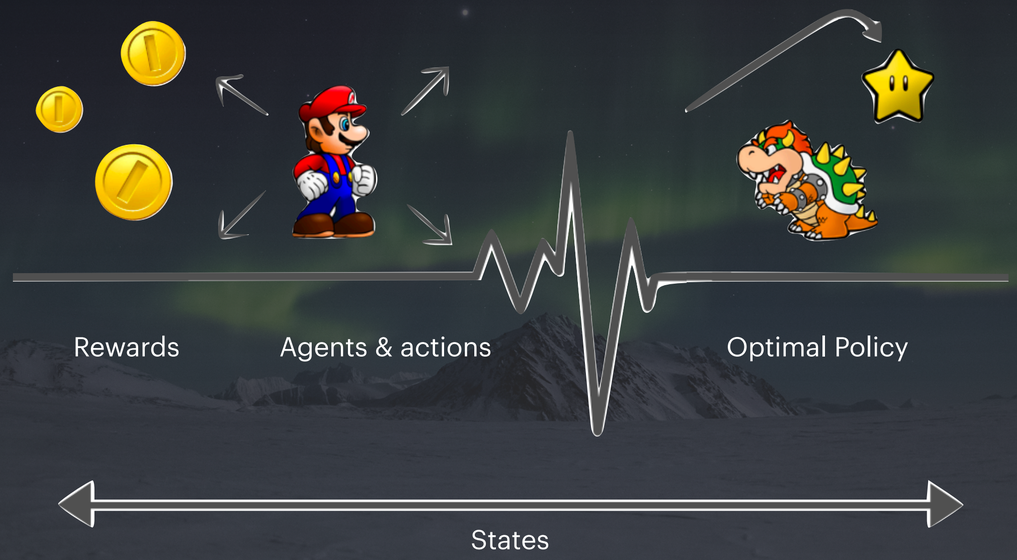
Environment Examples

Theory

Example: Cart-Pole

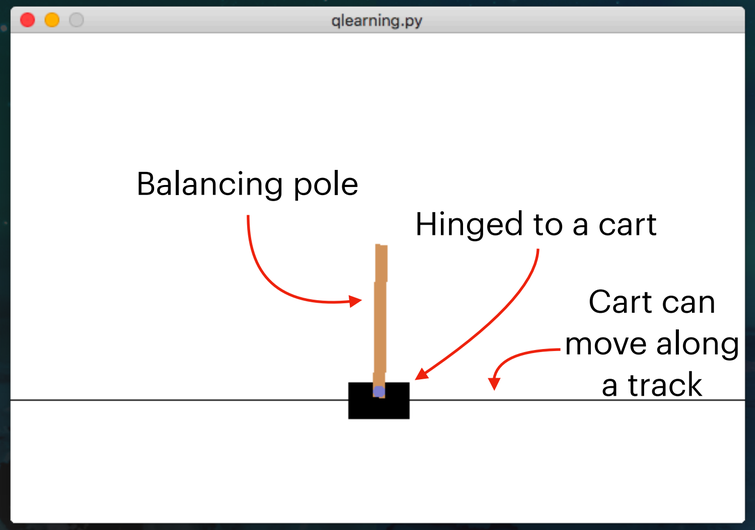


Example: Mario Bros



Application to Game Theory

Cart-Pole Environment



State space: $S = P_C \times V_C \times P_P \times V_P$

Possible actions: $A = \{a_L, a_R\}$

Reward: $u : S \to \mathbb{R}$

Infinitely Repeated Game

		Player 2: Cooperate	Player 2: Deviate
Player 1	Cooperate	4 , 4	1 , 6
	Deviate	6 , 1	2 , 2

State space: $S = \left[(A_1 \cup \phi) \times (A_2 \cup \phi)\right]^m$

Possible actions: $A_1 = \{a_{1,1}, a_{1,2}\}$

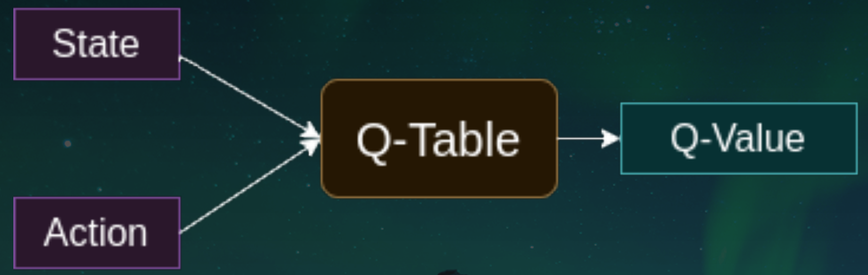
Reward: $u_1 : S \to \mathbb{R}$

What is Q-Learning?

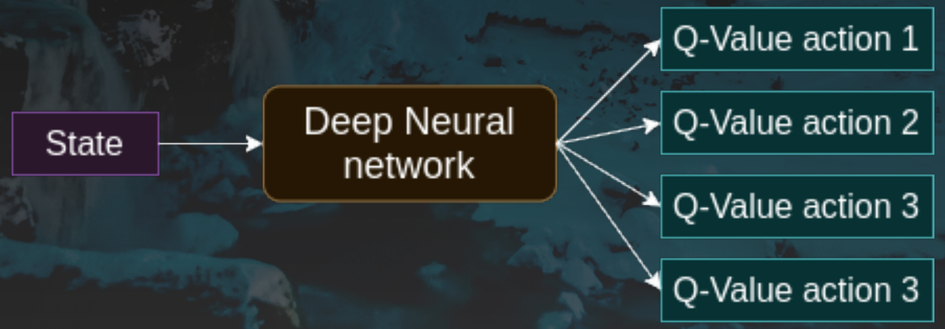
And why is it a good model for repeated games?

- **Model-free:** Does not require knowledge of state transitions and reward functions.
- **Value-based:** Learns a value function that maps states and actions to expected value.
- **Off-policy:** Actions are taken using a different policy than the value function, allowing the agent to explore.

Q-Learning

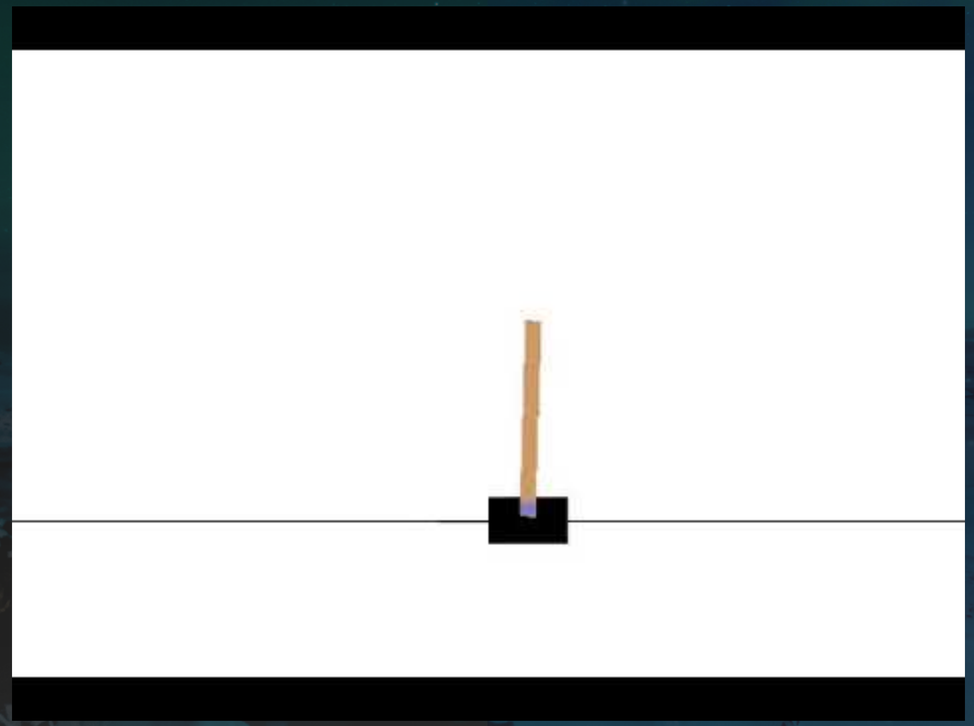


Deep Q-Learning



Progress Thus Far

Understanding Q-Learning



1. Implemented Deep Q-Network (DQN) algorithm using PyTorch
2. Tested the DQN implementation on Cart-Pole environment

Modeling Optimality in Python

```python
U_1 = np.array([[4, 1], [6, 2]])
U_2 = np.array([[4, 6], [1, 2]])
game = nash.Game(U_1, U_2)
print(game)
# Bi matrix game with payoff matrices:

# Row player:
# [[4 1]
#  [6 2]]

# Column player:
# [[4 6]
#  [1 2]]

nash_equilibria = game.support_enumeration()
print(list(nash_equilibria))  # [(array([0., 1.]), array([0., 1.]))]

```

1. Modeled a repeated game in Python and found optimal solution
2. Created an RL environment based off of the repeated game