Navigating Uncertainty

A Reinforcement Learning Approach to Solving the Frozen Lake Challenge

Qiutong Liu (liuqiu@kean.edu), Weixun Xie, Sihan Fu, Junyang Li, Zhirui Chen

with Dr. Israel R. Curbelo, Department of Mathematical Sciences, Kean University

Introduction

This project explores the application of Reinforcement Learning (RL) techniques to address the Frozen Lake environment from the OpenAI Gym, a popular platform for evaluating and developing RL algorithms. The Frozen Lake environment presents an agent with the task of navigating across a grid of ice and water tiles to reach a goal, without prior knowledge of the environment's dynamics. Our approach leverages the principles of Markov Decision Processes (MDPs) to model the environment, enabling the agent to learn optimal policies through interaction and feedback.

Methods

We implement the following two algorithms from scratch in Python using only Numpy (link to code under references.) NumPy is a library for doing math with arrays, which are like lists of numbers. It helps with tasks needed for making decisions in games or problems where you learn from experience.

Value Iteration (Update Rule):

$$V_{k+1}^{*}(s) = \max_{a}\left[\sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma V_{k}^{*}(s')\right)\right]$$

Q – Learning Algorithm:

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

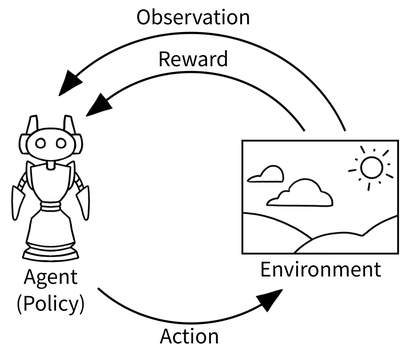
```
for s ∈ S, a ∈ A :
    Q[s, a] = 0
s = s0 // Or draw an s randomly from S
while True:
    a = select_action(s, Q)
    r, s' = execute(a)
    Q[s, a] = (1 − α)Q[s, a] + α(r + γ max_a' Q[s', a'])
    s = s'
```

Project Description

The game starts with the player at location [0,0] of the frozen lake grid world with the goal located at far extent of the world e.g. [3,3] for the 4x4 environment. Holes in the ice are distributed in random locations. The player makes moves until they reach the goal or fall in a hole. The lake is slippery meaning the player may move perpendicular to the intended direction sometimes.

Action Space

0: Left
1: Down
2: Right
3: Up



Observation Space

The observation represents the player's current position as

(current row) (number of rows) + current column

Rewards

Reach goal: +1
Reach hole: -0.1 (*custom*)
Reach frozen: 0

4x4 Example

Termination

The episode ends if:
1. The player moves into a hole
2. The player reaches the goal

Conclusions

Value Iteration always provides an optimal policy. The optimal policy for both the non-slippery and slippery version of the example provided are shown below.

```
[[1 2 1 0]
 [1 0 1 0]
 [2 1 1 0]
 [0 2 2 0]]
```

```
[[0 3 0 3]
 [0 0 0 0]
 [3 1 0 0]
 [0 2 1 0]]
```

Value Iteration requires that we have prior knowledge of the rewards and transition probabilities which usually isn't the case. However, we use these results to compare policies generated from Q-learning to an optimal policy.

Q-learning performs well without any prior knowledge of the rewards and transition probabilities. The policy generated for both the non-slippery and slippery version of the example provided are shown below.

```
[[1 2 1 0]
 [1 0 1 0]
 [2 1 1 0]
 [0 2 2 0]]
```

```
[[1 3 3 3]
 [0 0 2 0]
 [3 2 0 0]
 [0 2 2 0]]
```

Experimental results show that in an uncertain environment, the algorithm can learn effectively and reach the target position, which demonstrates the potential of reinforcement learning for solving problems.

References

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). The MIT Press.

https://github.com/OnlineDimension/ReinforcementLearning/blob/main/FrozenLake.py