Project 2: TD methods and n-step bootstrapping

Taariq Nazar
taariq.nazar@math.su.se

Instructions

All the solutions should be clearly explained and justified. Aim to be as short and concise as possible. Your submission should be typed (not handwritten) in the form of a single PDF containing your solutions, with code included as an appendix.

You are free to use any programming language of your choice to solve the coding parts of this project.

To pass this project, you need to score ≥ 50 points. The maximum score attainable is 100 points. The *Grading Criteria* document on the course page explains how the points of this project contribute to your final grade.

The project consists of practical exercises and are designed to give you deeper understanding of these topics. Take your time to fully understand how to solve these exercises!

TD-Methods & n-step bootstrapping

1. Suppose we have an environment with four states $\{s_0, s_1, s_2, s_4\}$. In state s_i, the agent can choose the actions $\{+, -\}$, where $(+)$ makes $s_i \to s_{i+1}$ and $(-)$, $s_i \to s_{i-1}$. Additionally, performing $(+)$ in s_4, takes the agent to s_0 and performing $(-)$ in s_0 takes the agent to s_4.

   Suppose you see the transition $s_0, (+) \to s_1, r = 2$.

   - Given the policy: $\pi(+) = 0.5$ and $\pi(-) = 0.5$, compute the one step update of $Q(s_0, +)$ using **Expected SARSA** Eq. (6.9) in the course book.
     Assume that $Q(s, a) = 1, ;\ \forall s, a$ before the update. Use the values $\alpha = 0.2$ and $\gamma = 0.9$.

2. An agent follows the behaviour policy b while collecting experience in an environment. However, the agent wants to evaluate the target policy π.

   Suppose you see the transition

   $$s_0, (+) \to s_1, (+) \to s_2. \tag{1}$$

   Where the policies are:

   - $\pi(+) = 0.5$ and $\pi(-) = 0.5$
   - $b(+) = 0.7$ and $b(-) = 0.3$

   Compute the importance sampling ratio $\rho_{0:1}$ for the trajectory given in (1).

Gridworld with a Monster

An agent is located in an $N \times N$ gridworld. The agent's goal is to **collect apples** while avoiding a **monster** that moves around the grid. The episode ends if the agent is caught by the monster or after a fixed number of time steps T.

At the beginning of each episode:

- The **agent**, **monster**, and **apple** spawn at random positions on the grid.

- The apple disappears once collected, and a **new apple appears randomly**.

The agent must navigate the grid to **maximize its total reward** by collecting apples while avoiding the monster.

Pseudo-code for simulating the environment is given below

State and Action Space

We model this problem as an MDP with the following components:

$$\mathcal{S} = \{(x_p, y_p), (x_m, y_m), (x_a, y_a) \mid x_p, y_p, x_m, y_m, x_a, y_a \in \{0, \ldots, N-1\}\} \tag{2}$$

$$\mathcal{A} = \{\text{left}, \text{up}, \text{right}, \text{down}\} \tag{3}$$

Where:

- (x_p, y_p) represents the agent's position.
- (x_m, y_m) represents the monster's position.
- (x_a, y_a) represents the apple's position.

Dynamics

The state dynamics follow these rules:

- **Agent Movement:**
  - The agent moves **deterministically** according to the chosen action.
  - If the action would take the agent outside the grid, it stays in place.
- **Monster Movement:**
  - The monster moves **randomly** in any direction with uniform probability. If it hits a wall, it stays in place.
- The monster moves when the agent moves.
- **Apple Respawn:**
  - If the agent collects the apple, a new apple spawns in a random empty cell (not occupied at the time when the apple is collected).
- **Terminal Conditions:**
  - The episode ends **immediately** if the agent is caught by the monster.
  - The episode also ends after T time steps.

Reward Function

The reward function is defined as:

$$R(s, a) = \begin{cases} +1 & \text{if the agent collects an apple} \\ -1 & \text{if the agent is caught by the monster} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Example Trajectory

Below is an example trajectory of the agent in the gridworld.

$$\begin{aligned}
&s_0 = ((2,2),(0,0),(3,3)), a_0 = \text{right}, r_1 = 0,\\
&s_1 = ((2,3),(1,0),(3,3)), a_1 = \text{up}, r_2 = 0,\\
&s_2 = ((1,3),(1,1),(3,3)), a_2 = \text{up}, r_3 = 0,\\
&s_3 = ((0,3),(1,2),(3,3)), a_3 = \text{right}, r_4 = 1,\\
&s_4 = ((0,3),(1,3),\text{new apple at }(2,0)), a_4 = \text{down}, r_5 = 0,\\
&s_5 = ((1,3),(0,3),(2,0)), a_5 = \text{left}, r_6 = -1,\\
&s_6 = ((1,3),(1,3),(2,0)).
\end{aligned} \tag{5}$$

At s_5, the agent is **caught by the monster**, ending the episode.

Tasks

Set size $N = 5$, episode length $T = 30$, discount factor $\gamma = 0.9$, and learning rate $\alpha = 0.1$. Initialize the Q-values to zero.

1. Implement the gridworld with a monster in code and implement the following methods using an ϵ-greedy policy:
   - SARSA
   - Q-Learning
   - Double Q-learning
   - n-step SARSA, for $n = 2, 4$
2. Plot the learning curve[1] for the different methods, a typical way to do this is to plot the total accumulated reward per episode over iteration, and answer the following questions:
   - Which method converges the fastest?
   - Is there an optimal n for n-step SARSA? Note that $n = 1$ is equivalent to SARSA.
3. Discuss the following:
   - Do the n-step SARSA methods converge at different rates? If yes, why. If no, why not?
   - Does double Q-learning differ from Q-learning in this case? Why or why not?
   - If the monster moves toward the player deterministically instead, how would this affect the convergence of the methods?

HINT: Experiment with different values for ϵ. You can also try a decaying ϵ.

NOTE: Adapt the pseudo-code below to implement the gridworld with a monster.

Pseudo-code for Simulating Gridworld with a Monster

```
S < - initialize()  # Initialize state (random player, monster, apple)
t < - 0  # Time step counter
terminal < - False
```

[1]The learning curve is a plot that monitors the agent's learning. You can determine a suitable learning curve on your own. A typical way to do this is to plot the total accumulated reward per episode (y) over iteration (x). See Figure (2) in page 132 of the course book for an example.

```
while not terminal and t < T:
    A < - \pi(S)  # Choose action based on policy \pi
    R < - 0  # Default reward

    S', R < - step(S, A)  # Take action and observe next state and reward

    # Check if player is caught by the monster
    if player_position in S' == monster_position in S':
        R < - -1
        terminal < - True  # End episode immediately

    # Check if player collects an apple
    if player_position in S' == apple_position in S':
        R < - 1
        apple_position < - random_empty_position()  # Spawn new apple

    # Update state and time step
    S < - S'
    t < - t + 1
```