Project 3: Approximation Methods and Policy Gradients

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!

Approximation Methods and Policy Gradients

Assume that we have the approximators $\hat{v}(s;\theta)$ and $\hat{q}(s,a;\theta)$ for the value function and action-value function, respectively. Both are parametrized by weights θ.

In approximation methods, we want to minimize the error between the true value (action-value) function and the approximated value function (action-value). We do this by iteratively updating the weights θ using stochastic gradients descent (SGD),

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha\delta_{t,v}\nabla_\theta \hat{v}(s;\theta_t), \text{ and} \\ \theta_{t+1} &= \theta_t + \alpha\delta_{t,q}\nabla_\theta \hat{q}(s,a;\theta_t). \end{aligned} \tag{1}$$

for some learning rate α. Here, δ is the estimation error, which is the difference between the true value and the approximated value, i.e. $\delta_{t,v} = v_\pi(s) - \hat{v}(s;\theta)$ and $\delta_{t,q} = q_\pi(s,a) - \hat{q}(s,a;\theta_t)$. See equations (9.4-9.5) in the course book.

In SGD we estimate $\delta_\cdot$. The estimate $\delta_\cdot$ depends on the method used, e.g SARSA, Q-Learning, Monte Carlo, etc. See the course book.

1. Denote, s as the state and $x(s)$ some feature of s. Assume that we have a linear approximator for the value function $\hat{v}(s;\theta) = \theta^T x(s)$. Show that the weights θ update according to
$$\theta_{t+1} = \theta_t + \alpha\delta_{t,v}x(s). \tag{2}$$
Hint: Use the defintion of update in (1).

2. For policy gradient methods, we update the policy weights according to
$$\theta_{t+1} = \theta_t + \alpha\delta_t\nabla_\theta \frac{\nabla_\theta \pi(a|s;\theta_t)}{\pi(a|s;\theta_t)} \tag{3}$$
see for instance Eq. (13.8) and (13.14) in the course book.

- Show that the following equality holds

$$\frac{\nabla_\theta \pi(a|s;\theta)}{\pi(a|s;\theta)} = \nabla_\theta \log \pi(a|s;\theta). \tag{4}$$

- Assume that our policy is a softmax-policy parameterized by θ, i.e

$$\pi(a|s;\theta) = \frac{e^{h(s,a;\theta)}}{\sum_b e^{h(s,b;\theta)}}, \tag{5}$$

with $h(s,a;\theta) = \theta^T x(s,a)$ for some feature function $x(s,a)$.
Show that the policy update in (3) is given by

$$\theta_{t+1} = \theta_t + \alpha\delta_t \left(x(s,a) - \sum_b \pi(b|s;\theta_t) x(s,b) \right). \tag{6}$$

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:

$$\mathcal{S} = \{(x_p, y_p), (x_m, y_m), (x_a, y_a) \mid x_p, y_p, x_m, y_m, x_a, y_a \in \{0, \ldots, N-1\}\} \tag{7}$$

$$\mathcal{A} = \{\text{left}, \text{up}, \text{right}, \text{down}\} \tag{8}$$

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 according to the chosen policy.
  - 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 or after T time step.

Reward Function

The reward function is defined as:

$$R(s,a) = \begin{cases} +1 & \text{if the agent collects an apple} \\ -1 & \text{if the agent is caught by the monster} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Example Trajectory

Below is an example trajectory of the agent in the 3×3 gridworld.

$$\begin{aligned} &s_0 = ((2,2),(0,0),(3,3)), a_0 = \text{right}, r_1 = 0, \\ &s_1 = ((3,2),(1,0),(3,3)), a_1 = \text{up}, r_2 = 0, \\ &s_2 = ((3,3),(1,1),(3,3)), a_2 = *, r_3 = 1, \\ &s_3 = ((3,3),(1,2),\text{new apple at } (2,0)), a_3 = \text{right}, r_4 = 0, \\ &s_4 = ((3,3),(2,2),(2,0)), a_4 = \text{down}, r_5 = 0, \\ &s_5 = ((3,2),(3,2),(2,0)), a_5 = \text{left}, r_6 = -1, \\ &s_6 = ((3,2),(3,2),(2,0)). \end{aligned} \tag{10}$$

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

Tasks

Set size $N = 10$, episode length $T = 200$, discount factor $\gamma = 0.95$.

Define the features

- $x(s) = (\Delta x \text{ to apple}, \Delta y \text{ to apple}, \Delta x \text{ to monster}, \Delta y \text{ to monster})^T$
- $\tilde{x}(s, a) = (\Delta x \text{ to apple}, \Delta y \text{ to apple}, \Delta x \text{ to monster}, \Delta y \text{ to monster}, a)^T$, where,

$$a = \begin{cases} (1,0,0,0) & \text{if left,} \\ (0,1,0,0) & \text{if up,} \\ (0,0,1,0) & \text{if right,} \\ (0,0,0,1) & \text{if down.} \end{cases} \tag{11}$$

For instance, if the state is $s = ((x_p, y_p), (x_m, y_m), (x_a, y_a))^T, a = \text{left}$, then the features are

- $x(s) = (x_a - x_p, y_a - y_p, x_m - x_p, y_m - y_p)^T$.
- $\tilde{x}(s, a) = (x_a - x_p, y_a - y_p, x_m - x_p, y_m - y_p, 1, 0, 0, 0)^T$.

Use the following linear approximators for the value function and action-value function:

- $\hat{v}(s; \phi) = \phi^T x(s)$
- $\hat{q}(s, a; \varphi) = \varphi^T \tilde{x}(s, a)$.

Use the following softmax-policy parameterized by θ,

$$\pi(a|s; \theta) = \frac{e^{h(s,a;\theta)}}{\sum_b e^{h(s,b;\theta)}}, \tag{12}$$

where $h(s, a; \theta) = \theta^T \tilde{x}(s, a)$.

1. Implement the following methods (start with learning rate $\alpha = 0.01$, but try different values):
   - Semi-gradient n-step SARSA, $n \in \{1, 2, 3\}$, page 247 in course book. Choose a suitable ϵ.
   - REINFORCE, page 328 in course book
   - REINFORCE with baseline, page 330 in course book
   - One step actor critic, page 332 in course book
2. Plot the learning curve[1] for the methods implemented above
3. Discuss the following:

[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.

- How does the learning rate α affect the learning speed and convergence of the methods?
- How do the methods above compare in terms of convergence speed?
- In this example, does policy gradient methods perform better than the value-based methods? Why or why not?

HINT: Use the results from above section.

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

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
```