Project 1: Dynamic Programming and Monte Carlo Methods

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!

Dynamic Programming

A cake production machine in a pastry factory can be in three different conditions: *pristine*, *worn* and *broken*. When operating the machine, it has a probability of θ of being degraded one step. That is,

$$\begin{aligned} p(s_{t+1} = \text{worn}|s_t = \text{pristine}, a_t = \text{continue }) = \theta \\ p(s_{t+1} = \text{broken}|s_t = \text{worn}, a_t = \text{continue }) = \theta. \end{aligned}$$

The factory owner can choose to *repair* the machine at a cost R or continue operating. If the machine is in pristine condition the factory owner earns C for producing exquisite cakes. If the machine is worn she earns $C/2$ for producing mediocre cakes. Furthermore, if the machine is broken, it will be replaced with a brand new pristine machine at a cost of Q.

This can be modeled as a Markov Decision Process (MDP) where the state and action space are

$$\begin{aligned} \mathcal{S} &= \{\text{pristine}, \text{worn}, \text{broken}\} \\ \mathcal{A} &= \{\text{continue}, \text{repair}\}, \end{aligned}$$

where the transition probabilities are given as

$$p(s_{t+1}|s_t, a_t) = \begin{cases} \text{If } a_t = \text{continue:} & \begin{bmatrix} 1-\theta & \theta & 0 \\ 0 & 1-\theta & \theta \\ 1 & 0 & 0 \end{bmatrix}, \\ \\ \text{If } a_t = \text{repair:} & \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \end{cases}$$

and the reward function is

$$R(s, a) = \begin{cases} \text{If } a = \text{continue:} & \begin{bmatrix} C \\ C/2 \\ -Q \end{bmatrix}, \\ \\ \text{If } a = \text{repair:} & \begin{bmatrix} -R \\ -R \\ -Q \end{bmatrix}. \end{cases}$$

Assume the parameters $\theta = 0.2, R = 5, C = 4, Q = 10$

1. Evaluate the *state value* function $v_\pi(s)$, for the policy $\pi(s) = \text{continue}; \ \forall s \in \mathcal{S}$. Present the value for each state.

   HINT: Look at equation (4.4) in the course book.

2. Perform one step of value iteration for the problem stated above.

3. Implement the above exercise in code and perform policy and value iteration using the discount factor $\gamma = 0.9$.

   - What is the optimal value and policy for each state? Answer with a table consisting of the rows: optimal value, optimal policy and states in the column.
   - Using different discount factors $\gamma = \{0.7, 0.8, 0.9\}$, compare the speed of convergence for value and policy iteration. That is, plot the **learning curve** for each discount factor.

   NOTE: You may get different values for the different discount factors. However, focus on how quickly the value/policy plateau.

   - Which setting of the discount factor converges the fastest? Comment on the different settings and give an explanation of the differences in convergence.

   HINT: Adapt the algorithms found in pages: 80 and 83 of the course book.

Monte Carlo Methods

An agent is located in a 4x4 gridworld. Starting at the bottom-left corner, the agents goal is to reach the top-right corner, see Figure 1.

The agent can move in four directions: up, down, left and right. If the agent hits a wall it will stay in the same position. To incentivize the agent to reach the goal as quickly as possible and not walk into walls, we provide the following rewards:

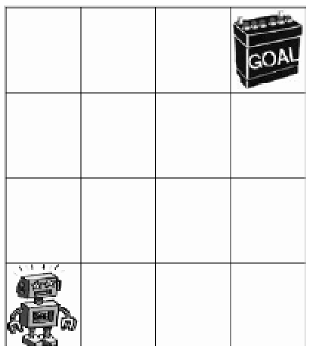

Figure 1: Image of a 4x4 gridworld. This image is taken from Johansson and Lansner [2012].

$$R(s,a) = \begin{cases} 10 & \text{if } s = \text{"Goal reached"} \\ -100 & \text{if } s = \text{"Wall hit"} \\ -1 & \text{if } s = \text{"Reaching non-goal state"}. \end{cases}$$

We can model this problem as an MDP where the state and action space are

$$\mathcal{S} = \{(0,0), \ldots (3,3)\}$$
$$\mathcal{A} = \{\text{left}, \text{up}, \text{right}, \text{down}\}\,.$$

Here $(0,0)$ denotes the bottom-left corner and $(3,3)$ denotes the top-right.

1. Given the following sequence of states and rewards,

$$\begin{aligned} s_0 &= (0,0), a_0 = \text{right}, r_1 = -1, \\ s_1 &= (0,1), a_1 = \text{right}, r_2 = -1, \\ s_2 &= (0,2), a_2 = \text{right}, r_3 = -1, \\ s_3 &= (0,3), a_3 = \text{up}, r_4 = -1, \\ s_4 &= (1,3), a_4 = \text{up}, r_5 = -1, \\ s_5 &= (2,3), a_5 = \text{up}, r_6 = -1, \\ s_6 &= (3,3), a_6 = \text{up}, r_7 = 10, \\ s_7 &= (3,3). \end{aligned}$$

   compute G_t for each time-step t.

2. Using the sequence above, perform one step of MC prediction. What is the estimated value for each state? Present your results in a 4x4 table representing the gridworld. The unvisited states should have a value of 0 or `nan`.

3. Implement the gridworld problem in code (adapt pseudo-code below). Use the discount factor $\gamma = 0.9$.

   - Estimate the optimal value $v^*(s)$ and policy $\pi^*(s)$ by implementing:
     - First-visit and every-visit MC with exploring starts.
     - Using ϵ-greedy policy, for $\epsilon = \{0.05, 0.1, 0.2\}$.
   - Compare the performance of the two methods above in terms of convergence and variance. Plot the learning curve for each method. Comment on the results of the different methods. Why do they differ? Is some method preferable over the other?

Pseudo-code for simulating gridworld

```
S < - (0,0) # Initialize state
while not terminal:
    A < - $\pi(S)$ # Choose action
    R < - -1 # Default reward
    S', R < - step(A) # Take action
    # Check if S' is outside gridworld, i.e walked into wall
    if S' is outside gridworld:
        R < - -100
        S' < - S
    if S' is goal:
        R < - 10

    # Estimate Value and policy here

    S,A,R,S' < - store(S,A,R,S') # Store state, action, reward and next state
```

References

Christopher Johansson and Anders Lansner. A Neural Reinforcement Learning System. 4 2012.