IT 567 - Lab Exercise 2
Winter 2024-2025

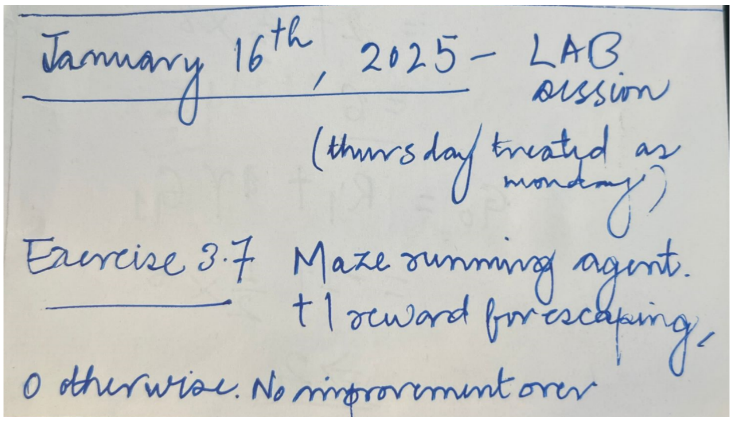
Kewalramani, Tanay
202201362

24 January 2025

Problem Set 1: Exercises 3.7, 3.8, and 3.9

Exercise 3.7: Insufficient Reward Signal

The agent struggles to improve its ability to escape the maze due to an insufficient reward signal. A sparse reward system, where +1 is given for escaping and 0 otherwise, does not effectively guide the agent because it provides no intermediate feedback.

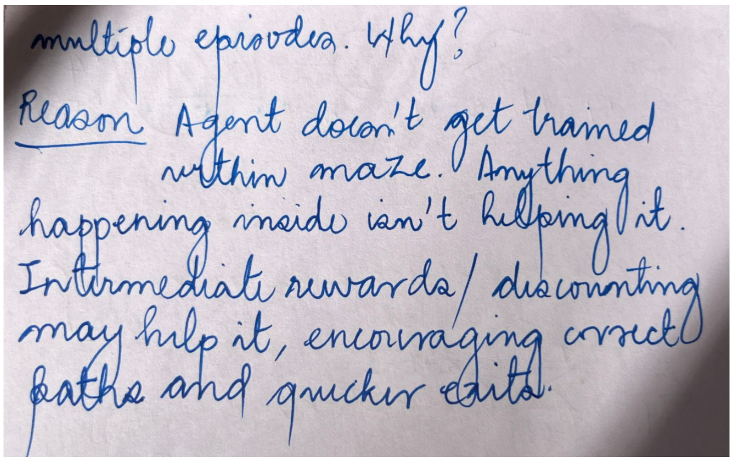


January 16th, 2025 - LAB session
(thursday treated as monday)

Exercise 3.7 Maze running agent.
+1 reward for escaping,
0 otherwise. No improvement over

Figure 1: First part of solution for exercise 3.7.

[H]



multiple episodes. Why?

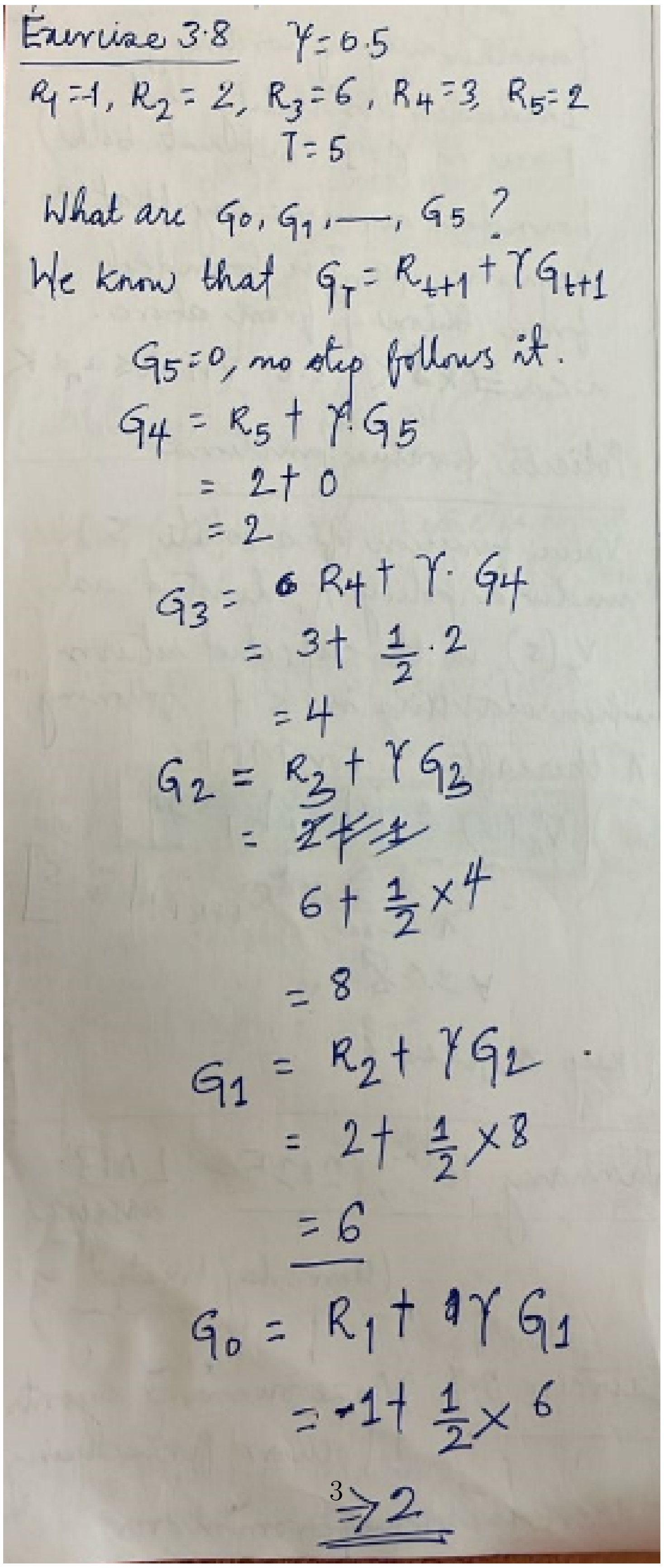
Reason Agent doesn't get trained within maze. Anything happening inside isn't helping it. Intermediate rewards / discounting may help it, encouraging correct paths and quicker exits.

Figure 2: Second part of solution for exercise 3.7.

Exercise 3.8: Calculating Returns Backward

Given $\gamma = 0.5$ and rewards $R_1 = -1, R_2 = 2, R_3 = 6, R_4 = 3, R_5 = 2$, with $T = 5$, the returns G_t are computed as:

$$G_t = R_t + \gamma G_{t+1}.$$



Exercise 3.8 $\gamma = 0.5$

$R_1 = -1,\ R_2 = 2,\ R_3 = 6,\ R_4 = 3\ \ R_5 = 2$

$T = 5$

What are $G_0, G_1, —, G_5$?

We know that $G_T = R_{t+1} + \gamma G_{t+1}$

$G_5 = 0$, no step follows it.

$$G_4 = R_5 + \gamma \cdot G_5$$
$$= 2 + 0$$
$$= 2$$

$$G_3 = \cancel{6}\ R_4 + \gamma \cdot G_4$$
$$= 3 + \frac{1}{2} \cdot 2$$
$$= 4$$

$$G_2 = R_3 + \gamma G_3$$
$$= \cancel{2 + 1}$$
$$6 + \frac{1}{2} \times 4$$
$$= 8$$

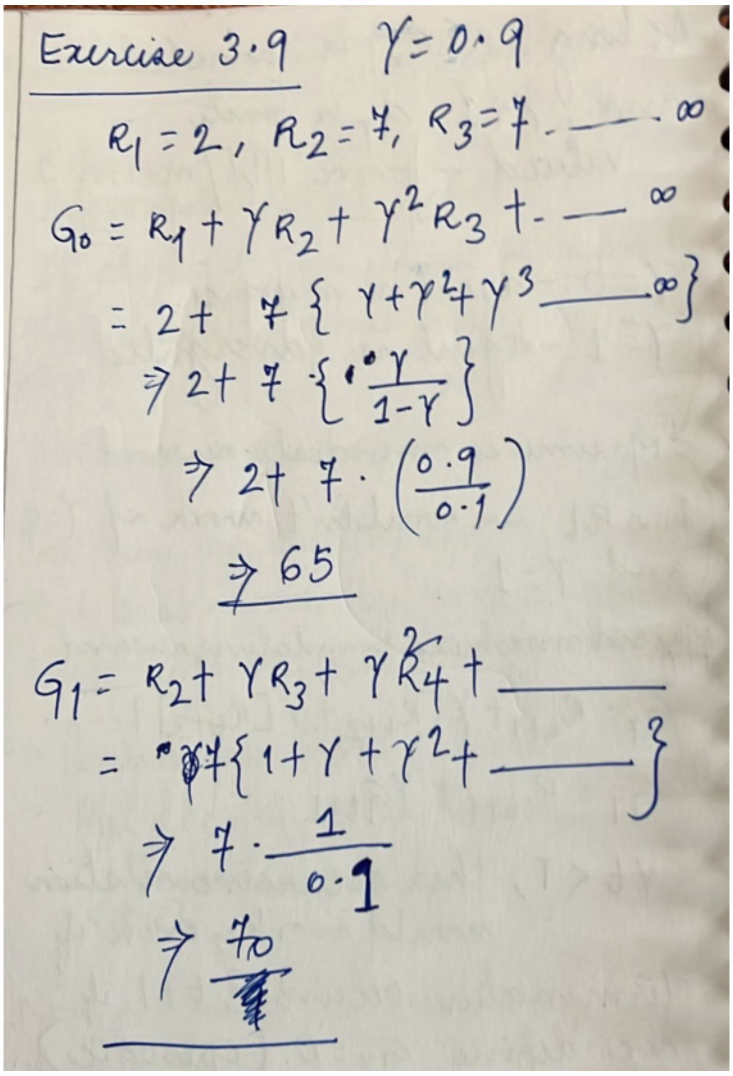
$$G_1 = R_2 + \gamma G_2$$
$$= 2 + \frac{1}{2} \times 8$$
$$= 6$$

$$G_0 = R_1 + \gamma G_1$$
$$= -1 + \frac{1}{2} \times 6$$
$$\Rightarrow 2$$

Figure 3: Solution for exercise 3.8.

Exercise 3.9: Infinite Rewards with Discount Factor

For $\gamma = 0.9$ and rewards $R_1 = 2$ followed by an infinite sequence of 7s, calculate G_t.



Exercise 3.9 $\gamma = 0.9$

$R_1 = 2,\ R_2 = 7,\ R_3 = 7 \ldots\ldots \infty$

$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots \infty$

$= 2 + 7\{\gamma + \gamma^2 + \gamma^3 \ldots\ldots \infty\}$

$\Rightarrow 2 + 7 \cdot \left\{\frac{\gamma}{1-\gamma}\right\}$

$\Rightarrow 2 + 7 \cdot \left(\frac{0.9}{0.1}\right)$

$\Rightarrow 65$

$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \ldots$

$= \cancel{\gamma}7\{1 + \gamma + \gamma^2 + \ldots\}$

$\Rightarrow 7 \cdot \frac{1}{0.1}$

$\Rightarrow 70$

Figure 4: Second part of solution for exercise 3.7.

Problem Set 2: Exercise 3.6

Return at Failure

For an episodic task, the return is:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

At failure (step K):

$$G_t = -\gamma^K,$$

where K is the time to failure.

Continuing Task

For a continuing task, the reward is -1 only on failure, and the return is computed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Comparison:

- Episodic Task: $G_t = -\gamma^K$, explicitly dependent on K.
- Continuing Task: Failure contributes $-\gamma^K$ as part of a longer time horizon.

Problem Set 3: Exercises 3.14, 3.15, and 3.16

Before we begin with the three exercises in this set, we're asked to simulate the gridworld problem on python.

Following is the python code for the Gridworld problem:

Code:

```
import numpy as np
import gym
from gym import spaces

class GridworldEnv(gym.Env):
    def __init__(self, grid_size=5, gamma=0.9):
        super(GridworldEnv, self).__init__()

        self.grid_size = grid_size
        self.gamma = gamma

        # Define the action and observation space
        # Actions: 0 = North, 1 = South, 2 = East, 3 =
            West
        self.action_space = spaces.Discrete(4)
        # Observations: Grid positions (x, y)
        self.observation_space = spaces.Tuple((
            spaces.Discrete(grid_size), spaces.Discrete
                (grid_size)
        ))

        # Define special states and their rewards
        self.special_states = {
            (0, 1): (4, 1, 10),  # Transition to (4, 1)
                with reward 10
            (0, 3): (2, 3, 5),   # Transition to (2, 3)
                with reward 5
        }

        # Initialize grid and agent position
        self.grid = np.zeros((grid_size, grid_size))
        self.reset()

    def reset(self):
        self.agent_pos = (np.random.randint(0, self.
            grid_size), np.random.randint(0, self.
            grid_size))
        return self.agent_pos

    def step(self, action):
        x, y = self.agent_pos

        if self.agent_pos in self.special_states:
            next_x, next_y, reward = self.
                special_states[self.agent_pos]
        else:
            if action == 0:  # North
```

```
                next_x, next_y = max(x - 1, 0), y
            elif action == 1:  # South
                next_x, next_y = min(x + 1, self.
                    grid_size - 1), y
            elif action == 2:  # East
                next_x, next_y = x, min(y + 1, self.
                    grid_size - 1)
            elif action == 3:  # West
                next_x, next_y = x, max(y - 1, 0)

            reward = 0 if (next_x, next_y) != (x, y)
                else -1

        self.agent_pos = (next_x, next_y)
        return self.agent_pos, reward, False, {}

    def render(self):
        grid = np.zeros((self.grid_size, self.grid_size
            ), dtype=str)
        grid[:] = '.'
        x, y = self.agent_pos
        grid[x, y] = 'A'
        print("\n".join([" ".join(row) for row in grid
            ]))
        print()

# Create Gridworld environment
env = GridworldEnv()

# Initialize value function
v_pi = np.zeros((env.grid_size, env.grid_size))

# Perform policy evaluation for uniform random policy
policy = 0.25  # Equal probability for all four actions
threshold = 1e-4
delta = float('inf')

while delta > threshold:
    delta = 0
    for x in range(env.grid_size):
        for y in range(env.grid_size):
                v = v_pi[x, y]
                new_value = 0
                for action in range(env.action_space.n):
                    env.agent_pos = (x, y)
                    next_state, reward, _, _ = env.step(
                        action)
                    nx, ny = next_state
                    new_value += policy * (reward + env.
                        gamma * v_pi[nx, ny])
                v_pi[x, y] = new_value
                delta = max(delta, abs(v - v_pi[x, y]))

# Print final value function
print("Value Function for Uniform Policy:")
print(np.round(v_pi, 2))
```

[a4paper,12pt]article amsmath,amssymb,graphicx,hyperref
IT 567 - Lab Exercise 2
Winter 2024-2025 Kewalramani, Tanay
202201362 24 January 2025

Problem Set 3: Exercises 3.14, 3.15, and 3.16

Exercise 3.14: Bellman Equation

The Bellman equation for a state s is:

$$v^{\pi}(s) = \sum_{s'} P(s' \mid s, a)\left[R(s, a, s') + \gamma v^{\pi}(s')\right].$$

For the center state, $v(s) = 0.7$, and its four neighboring states have values $2.3, 0.4, -0.4, 0.7$, with equal transition probabilities to each neighboring state.

Solution: Using the Bellman equation for equiprobable transitions:

$$v(s) = \frac{1}{4}\left[R + \gamma(2.3) + R + \gamma(0.4) + R + \gamma(-0.4) + R + \gamma(0.7)\right].$$

Simplifying:

$$v(s) = R + \frac{\gamma}{4}\left(2.3 + 0.4 - 0.4 + 0.7\right).$$

Substitute $\gamma = 0.9$ and $v(s) = 0.7$:

$$0.7 = R + \frac{0.9}{4} \cdot 3.0.$$

$$0.7 = R + 0.675.$$

Solve for R:

$$R = 0.7 - 0.675 = 0.025.$$

Answer: The reward $R = 0.025$ ensures the Bellman equation holds for $v(s) = 0.7$.

—

Exercise 3.15: Impact of Reward Signs

In the gridworld example: - Positive Rewards: Represent desirable states (e.g., goal states). - Negative Rewards: Represent undesirable states (e.g., falling off the grid). - Zero Rewards: Represent neutral states.

Solution: The agent optimizes its policy based on the relative differences between rewards, not their absolute values. To illustrate this, consider adding a constant c to all rewards. Let $v^\pi(s)$ be the value function for state s, defined as:

$$v^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid s_t = s \right].$$

Adding c to all rewards:

$$R'_{t+1} = R_{t+1} + c.$$

The new value function $v'^\pi(s)$ becomes:

$$v'^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t (R_{t+1} + c) \mid s_t = s \right].$$

Expanding:

$$v'^\pi(s) = v^\pi(s) + \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t c \mid s_t = s \right].$$

Since c is constant:

$$\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}.$$

Thus:

$$v'^{\pi}(s) = v^{\pi}(s) + \frac{c}{1-\gamma}.$$

Conclusion: Adding c shifts all values by $\frac{c}{1-\gamma}$, but it does not affect the relative differences between states. Therefore, the agent's policy remains unchanged.

—

Exercise 3.16: Modified Bellman Equation

Adding a constant c to all rewards modifies the Bellman equation. Let the original Bellman equation be:

$$v^{\pi}(s) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma v^{\pi}(s') \right].$$

When c is added to all rewards:

$$R'(s, a, s') = R(s, a, s') + c.$$

The modified Bellman equation becomes:

$$v'^{\pi}(s) = \sum_{s'} P(s' \mid s, a) \left[(R(s, a, s') + c) + \gamma v'^{\pi}(s') \right].$$

Expanding:

$$v'^{\pi}(s) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma v'^{\pi}(s') \right] + \sum_{s'} P(s' \mid s, a) c.$$

Since $\sum_{s'} P(s' \mid s, a) = 1$:

$$v'^{\pi}(s) = v^{\pi}(s) + c.$$

Conclusion: Adding c shifts the value function by a constant c. The policy remains unaffected because the relative values of states are preserved.

—