National University of Singapore
School of Computing

Semester 1, AY2023-24

CS4246/CS5446 AI Planning and Decision Making

---

Due: 17 Nov 2023 (Soft); 24 Nov 2023@23:59 (Hard)

**Assignment 3**

# Instructions

- This assignment requires 2 submissions.
  - Submit the PDF file containing your solutions to this assignment on Canvas.
  - You can use this Overleaf project to write your solutions.
  - Submit the ZIP file containing your code to this separate assignment on Canvas.
- You have only one attempt on Canvas.
- You are required to specify your group details in your attempt (group number can be found on Canvas) by updating the `team.tex` file with the relevant details.
- **Total marks: 10; Weightage 10% of final marks**
- On collaboration:
  - The goal of the assignment is to understand and apply the concepts in the class.
  - You may discuss the assignment with other groups via the discussion forum.
  - It is OK for the solution ideas to arise out of such discussions. However, it is considered plagiarism if the solution submitted is highly similar to other submissions or to other sources.
- Citing help and reference
  - At the end of the assignment, clearly cite the sources you referred to when arriving at the answer (Books, External notes, Generative AI tools, etc.,).

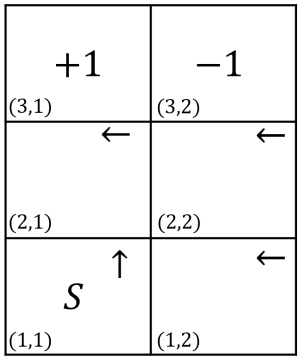
# Team members

**Group number:** A35
**Member 1 details:**
- Name: Niharika Shrivastava
- NUSNet id: E0954756
- Student number: A0254355A

**Member 2 details:**
- Name: Harshavardhan Abichandani
- NUSNet id: E0945792
- Student number: A0250610X

# 1 Reinforcement Learning

[3 marks] Consider the environment shown in Figure 1 (left). The agent can execute actions to move in the four cardinal directions. The policy the agent follows is indicated using the arrows in the states. Note that the reward for the states other than $(3, 1)$ and $(3, 2)$ is zero. The agent runs the policy to obtain the trails as shown in Figure 1 (right).




Trial 1: (1,1) → (2,1) → (3,1)
Trial 2: (1,1) → (2,1) → (2,2) → (2,1) → (3, 1)
Trial 3: (1,1) → (2,1) → (2,2) → (1,2) → (2,2) → (3,2)
Trial 4: (1,1) → (1,2) → (2,2) → (3,2)

Figure 1: Environment and Trials

1. (1 mark) Compute the transition function for the transitions you can observe from the trials.

   Solution:

| State ($s$) | Action ($a$) | Next State ($s'$) | $P(s'\|s, a)$ |
|---|---|---|---|
| (1, 1) | Up | (2, 1) | 0.75 |
| (1, 1) | Up | (1, 2) | 0.25 |
| (1, 2) | Left | (2, 2) | 1 |
| (2, 1) | Left | (3, 1) | 0.5 |
| (2, 1) | Left | (2, 2) | 0.5 |
| (2, 2) | Left | (2, 1) | 0.25 |
| (2, 2) | Left | (1, 2) | 0.25 |
| (2, 2) | Left | (3, 2) | 0.5 |

   All other transitions have probability 0.

2. (2 marks) Describe the problem with ADP based on your observation from computing the transition function from the trials in the above.

   Solution: This is a passive ADP, i.e., the policy is fixed and we learn the transition function empirically through experience - to finally get the utility function of the environment.

   (a) From the above computation, we observe that the learnt transition function is highly inaccurate for the given policy. For instance, $P(s' = (2, 2)|s = (1, 2), a = Left) = 1$.

Even if the environment is considered to be stochastic, this learnt transition will never align with the true transition intended by the policy ($P(s' = (1,1)|s = (1,2), a = Left)$). Thus, we need a lot more trials and data to learn the true transition function.

(b) Consequently, if the learnt transition function is inaccurate, the learnt utility function will be inaccurate. Therefore, ADP fails when the number of trials/observations is too low and not representative of the actual environment,

# 2 Game Theory

[2 marks] Both Row player ($P1$) and Column player ($P2$) have three options $A, B, C$ to choose. The payoff matrix for the combination of choices they make is as shown below

| | | P2 | | |
|---|---|---|---|---|
| | | A | B | C |
| P1 | A | -5, -5 | 0, 5 | 5, 10 |
| | B | 5, 0 | -10, -10 | -2.5, -7.5 |
| | C | 10, 5 | 7.5, -2.5 | -15, -15 |

1. (2 mark) Compute the pure strategy Nash equilibria and write the strategy profile(s) for the game.

   Solution:

   By fixing a strategy for P1 to A, P2 can not switch from strategy C to any other to get more reward. Symmetric for the other case.

   Pure strategy Nash Equilibria: (C, A) = (10, 5) and (A, C) = (5, 10).
   Strategy profile:

   - P1: A, P2: C
   - P1: C, P2: A

# 3 Programming assignment

1. (2 marks) Briefly state and explain the algorithm you chose, hyperparams used (if any) and your observation/learning as you implemented the solution. Keep your description succinct. Do not write more than 1 page for this part! You are not required to give the algorithm pseudocode or detailed description.

Write your answer here:

Deep Q Network (DQN) [2] can be only implemented for discrete action because of the $\max Q$ step.Therefore, we used it for our elevator environment. It is an off-policy method that tries to learn the $Q$ values by using a function approximator. In our case, the function approximator is a simple Deep Neural Network (DNN). The error term of the DQN comes directly from the bellman update equation.

$$Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a') \tag{1}$$

Here the value of the state-action pair depends on the immediate reward and the max value of the next state. DQN simply reduces the difference between the two terms given below. In equation 2 our goal is to update weights so that our current estimate is closer to the target network.

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (r_t^i + \gamma \max_{a'} Q_{\bar{w}}(s_{t+1}^i, a') - Q_w(s_t^i, a_t^i))^2 \tag{2}$$

The traditional implementation of DQN involves updating the target network $Q_\pi^t$ after a fixed number of steps. This is done to make the training of DQN stable. We utilize a different method, we soft update the target network at every step. Based on [3], this way is better and is relatively more stable.

| Hyperparameters | $\gamma$ | $\epsilon_{start}$ | $\epsilon_{end}$ | $\epsilon_{decay}$ | $\tau$ | $batch - size$ |
|---|---|---|---|---|---|---|
| Values | 0.99 | 0.9 | 0.05 | 1000 | 0.005 | 128 |

where,

- $batch - size$ is the number of transitions sampled from the replay buffer.
- $\gamma$ is the discount factor.
- $\epsilon_{start}$, $\epsilon_{end}$ is the start and end values of the exploration factor.
- $\epsilon_{decay}$ controls the rate of exponential decay of epsilon, higher means a slower decay.
- $\tau$ is the update rate of the target network.

We had to heavily tune the hyperparameters in order to get the desired rewards of $600+$. However, with the right set of hyperparameters, DQN converged towards desired rewards in about $1000$ episodes. Initially, we had used PPO which gave us a maximum expected reward of only $400$. However, DQN took much less time to train than PPO.

# References

[1] https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2019.