CS 486/686 Assignment 4 (64 marks)

Alice Gao

Due Date: 11:59 PM ET on Tuesday, December 7, 2021 with an extension with no penalty to 11:59 pm ET on Wednesday, December 15, 2021

Changes

- v1.4 In `utils.py`, `episode_with_greedy_policy()` was updated to ensure proper calculation of the discounted reward. Specifically, the exponent to which the discount factor is raised is now correct. Note that this function is not required to pass any unit tests; it exists to help you assess your learned Q-values.
- v1.3 Updated the pseudocode to indicate where $N(s, a)$ should be updated.
- v1.2 To ensure that test results are reproducible, we've left one random sampling function `utils.py`. All random sampling should be done with `select_integer_from_categorical_distribution(P)`.
- v1.1 Added images in the `code_posted/img/` folder.

Academic Integrity Statement

If your written submission on Learn does not include this academic integrity statement with your signature (typed name), we will deduct 5 marks from your final assignment mark.

I declare the following statements to be true:

- The work I submit here is entirely my own.
- I have not shared and will not share any of my code with anyone at any point.
- I have not posted and will not post my code on any public or private forum or website.
- I have not discussed and will not discuss the contents of this assessment with anyone at any point.
- I have not posted and will not post the contents of this assessment and its solutions on any public or private forum or website.
- I will not search for assessment solutions online.
- I am aware that misconduct related to assessments can result in significant penalties, possibly including failure in the course and suspension. This is covered in Policy 71: https://uwaterloo.ca/secretariat/policies-procedures-guidelines/policy-71.

Failure to accept the integrity policy will result in your assignment not being graded.

By typing or writing my full legal name below, I confirm that I have read and understood the academic integrity statement above.

Instructions

- Submit the assignment in the A4 Dropbox on Learn. No late assignment will be accepted. This assignment is to be done individually.
- I strongly encourage you to complete your write-up in Latex, using this source file. If you do, in your submission, please replace the author with your name and student number. Please also remove the due date, the Instructions section, and the Learning goals section.
- Lead TAs:
  - Ji Xin
  - Blake VanBerlo

  The TAs' office hours will be posted on MS Teams.

Learning goals

Decision Networks

- Model a real-world problem as a decision network with sequential decisions.
- Given a decision network with sequential decisions, determine the optimal policy and the expected utility of the optimal policy by applying the variable elimination algorithm.

Reinforcement Learning

- Implement the active adaptive dynamic programming algorithm for reinforcement learning.

1 Decision Network for "Monty Hall" (28 marks)

The Monty Hall Problem is stated as follows.

You are on a game show, and you are given the choice of three doors: Behind one door is a car; behind the others, goats. The host knows what's behind each door but you don't.

1. First, you pick a door, say Door 1.
2. Second, the host opens another door, say Door 3, which has a goat behind it.
3. Finally, the host says to you, "Do you want to pick Door 2?"

Is it to your advantage to switch your choice in step 3?

In step 2, the host always opens a door with a goat behind it, but not the door you chose first, regardless of which door you chose first. If both remaining doors have goats behind them, the host will open a door randomly. In step 3, you "reserve" the right to open the door you chose first, but can change to the remaining door after the host opens the door to reveal a goat. Your price is the item behind the final door you choose. You prefer cars over goats (cars are worth 1 and goats are worth 0). The car is behind doors 1, 2, and 3 with probabilities p_1, p_2 and $1 - p_1 - p_2$ respectively, and you know the values of p_1 and p_2.

Let's model this problem using the following variables.

- CarDoor $\in \{1, 2, 3\}$ is the door such that the car is behind it. This is a random variable.
- FirstChoice $\in \{1, 2, 3\}$ is the index of the door you picked first. This is a decision variable.
- HostChoice $\in \{smaller, bigger\}$ is the index of the door picked by the host. The value of this variable indicates whether the index of the door picked is the smaller or bigger one of the two doors left after you made your first choice. This is a random variable.
- SecondChoice $\in \{stay, switch\}$ indicates whether you stay with the door you picked first or switch to the remaining door not opened by the host. This is a decision variable.
- Utility $\in \{0, 1\}$ is 0 if you get a goat and 1 if you get a car.

Please complete the following tasks.

(a) Complete the decision network in Figure 1 by drawing all the arcs. Show the probability table for each random variable. Show the utility table for the utility node. You can use p_1 and p_2 in your tables since you do not know their values yet.

Hint: When you are deciding whether node A should be a parent of node B (a decision variable), think of the following. If node A is a parent of node B, then the optimal policy may be of a form that: if A has value a, then B should be b. Otherwise, if node A is not a parent of node B, the optimal policy for B cannot depend on the value of A. In other worlds, adding an edge in the network increases the set of possible policies that we would consider.

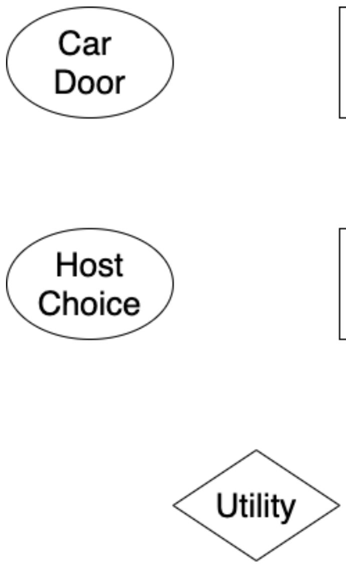

Figure 1: The Monty Hall Problem

Marking Scheme:

(10 marks)

- (4 marks) Correct parent nodes.
- (3 marks) Correct probability tables.
- (3 marks) Correct utility table.

(b) **Assume that $p_1 = 1/3$ and $p_2 = 1/3$.**

Compute the optimal policy for the decision network by applying the variable elimination algorithm. Show all your work including all the intermediate factors created. Clearly indicate the optimal policy and the expected utility of the agent following the optimal policy.

Marking Scheme:

(9 marks)

- (4 marks) Sum out variables in the correct order.
- (2 marks) Correct optimal policy for FirstChoice.
- (2 marks) Correct optimal policy for SecondChoice.
- (1 mark) Correct value of the expected utility of the optimal policy.

(c) Consider a different case where $p_1 = 0.7$ **and** $p_2 = 0.2$**.** The car is much more likely to be behind door 1 than to be behind door 2 or 3. The car is slightly more likely to be behind door 2 than to be behind door 3.

Compute the optimal policy for the decision network by using the variable elimination algorithm. Show all your work including all the intermediate factors created. Clearly indicate the optimal policy and the expected utility of the agent following the optimal policy.

Marking Scheme:

(9 marks)

- (4 marks) Sum out variables in the correct order.
- (2 marks) Correct optimal policy for FirstChoice.
- (2 marks) Correct optimal policy for SecondChoice.
- (1 mark) Correct value of the expected utility of the optimal policy.

2 Reinforcement Learning (36 marks)

You will explore the wumpus world using the **Q-learning** and SARSA algorithms.

Section 2.1 describes our version of the wumpus world. Section 2.2 describes the Q-learning and SARSA algorithms. Section 2.3 describes action selection strategies that manage the exploration/exploitation trade-off.

We have provided three Python files. Please read the comments in the provided files carefully.

1. `wumpus_env.py`

   contains a `WumpusWorld` class that simulates the dynamics of the wumpus world. It implements the OpenAI gym interface[1]. **Do not change this file.**

2. `utils.py`

   contains functions for random sampling from various distributions. To pass the unit tests, **use these functions for ALL random sampling in your implementation. Do not change this file.**

3. `rl.py`

   implement all the empty functions except for `select_action()`. You will use `select_action()` to sample actions. **Do not change any function signatures.**

Please complete the following tasks.

1. Implement all the empty functions in `rl.py`. Submit `rl.py` only on Marmoset.

   Marking Scheme: (30 marks)

   Unit tests

   - `select_action_epsilon_greedy`
     (1 public test + 2 secret tests) * 1 mark = 3 marks
   - `select_action_softmax`
     (1 public test + 2 secret tests) * 2 marks = 6 marks
   - `select_action_optimistically`
     (1 public test + 2 secret tests) * 1 mark = 3 marks

[1]OpenAI gym environments are commonly used by researchers as benchmarks to assess the performance of reinforcement learning methods.

- `active_q_learning`
  (1 public test + 2 secret tests) * 3 marks = 9 marks
- `active_sarsa`
  (1 public test + 2 secret tests) * 3 marks = 9 marks

2. You will investigate the performance of Q-learning and SARSA with different exploration strategies on the Wumpus World environment.

Produce four plots for the following combinations.

- Q-learning + softmax exploration with $T = 0.1$
- SARSA + softmax exploration with $T = 0.1$
- Q-learning + optimistic utility exploration with $N_e = 2, R^+ = 999$
- SARSA + optimistic utility exploration with $N_e = 2, R^+ = 999$

Use the learning rate $\alpha = 0.5$ and discount factor $\gamma = 0.99$. Instantiate the Wumpus World environment with the default constructor arguments.

Run the program 3 times. Each time, run the program for $50,000$ episodes. For each run, use a different random seed by calling `WumpusWorld.seed(s)` prior to learning. In each splot, the x-axis denotes the episodes, and the y-axis denotes the total undiscounted reward obtained in each episode averaged over the 3 runs.

To make the plot smoother, we will plot each value to be the moving average of the previous 50 episodes. A moving average is defined as the average of the last N elements in a series, where N is the window size. You can use the Python command below to calculate a moving average of a 1D numpy array.

```
rewards_mov_avg = np.convolve(rewards, np.ones(N)/N, mode=`valid')
```

We have included an example plot in Figure 2 depicting the performance of Q-learning with ϵ-greedy exploration ($\epsilon = 0.1$).

Marking Scheme: (4 marks)

- (1 mark) Plot for Q-learning with softmax exploration
- (1 mark) Plot for SARSA with softmax exploration
- (1 mark) Plot for Q-learning with optimistic utility estimates
- (1 mark) Plot for SARSA with optimistic utility estimates

2. If you were to explore the wumpus world, which learning algorithm and exploration strategy would you choose among the options below?

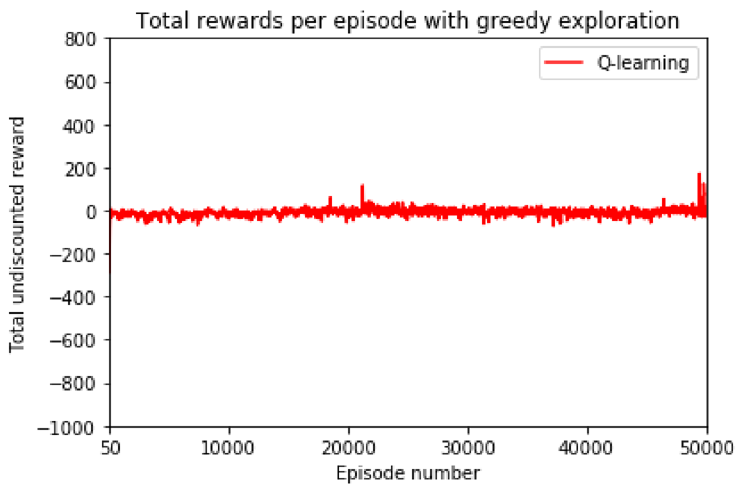

Figure 2: Q-learning with ϵ-greedy action selection ($\epsilon = 0.1$) averaged across 3 runs, smoothed with a moving average ($N = 50$).

- Q-learning v.s. SARSA
- softmax v.s. optimistic utility

Provide your answer and justify it with no more than 3 sentences. You can base your answer on the previous parts or perform more experiments yourself.

Marking Scheme: (2 marks) A reasonable justification.

2.1 The Wumpus World

The following description is adapted from Artificial Intelligence: A Modern Approach by Russell and Norvig. Check section 7.2 on page 236 in the 3rd edition or section 7.2 on page 210 in the 4th edition. The only difference between our version and the textbook version is that the three movement actions (`Forward`, `TurnLeft`, `TurnRight`) has no effect with a small probability κ.

The wumpus world is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the terrible wumpus, a beast that eats anyone who enters its room. The agent can shoot the wumpus, but the agent has only one arrow. Some rooms contain bottomless pits that will trap the agent if the agent wonders into these rooms. The only redeeming feature of this bleak environment is the possibility of finding a heap of gold.

The agent takes one of the following actions during each time step:

- `Forward`: This action is noisy with an error probability of $\kappa \in [0, 1]$. With a probability of $1-\kappa$, the agent moves to the next square in the direction the agent is facing. If the agent bumps into a wall, it remains in the same square. With a probability of κ, the action has no effect and the agent remains in the same square.

- `TurnLeft`: This action is noisy with an error probability of $\kappa \in [0, 1]$. With a probability of $1-\kappa$, the agent rotates 90 degrees to the **left (counterclockwise)** and remains in the same square. With a probability of κ, the action has no effect and the agent faces the same direction as before.
- `TurnRight`: This action is noisy with an error probability of $\kappa \in [0, 1]$. With a probability of $1-\kappa$, the agent rotates 90 degrees to the **right (clockwise)** and remains in the same square. With a probability of κ, the action has no effect and the agent faces the same direction as before.
- `Grab`: The agent obtains the gold if the gold is in the same square as the agent. Otherwise, nothing happens.
- `Shoot`: The agent fires an arrow in a straight line in the direction the agent is facing. The arrow continues until it hits and kills a wumpus or hits a wall. The agent has one arrow. If the agent has already used the arrow, this action has no effect.
- `Climb`: The agent climbs out of the cave if it is at the entrance. Otherwise, this action has no effect. Once the agent climbs out of the cave, this episode of the wumpus world journey ends.

The agent may obtain the following rewards.

- $+1000$ for climbing out of the cave with the gold.
- -1000 for falling into a pit.
- -1000 for being eaten by the wumpus.
- -1 for each action taken (even if the action has no effect).
- -10 for using up the arrow.

In addition to knowing its current location and orientation, the agent may receive the following observations:

- In the squares directly (not diagonally) adjacent to the wumpus, the agent will perceive a `stench`.
- In the squares directly (not diagonally) adjacent to a pit, the agent will perceive a `breeze`.

- In the square where the gold is, the agent will perceive a `glitter`.
- When the agent bumps into a wall, it will perceive a `bump`.
- When the wumpus is killed, it emits a woeful `scream` that can be perceived anywhere in the cave.

2.2 The Active Q-Learning and SARSA Algorithms

Let's define some quantities that we will use in both algorithms.

- γ: the discount factor in $(0, 1]$.
- $R(s)$: the immediate reward of entering any state.
- $Q(s, a)$: the Q-values for each state-action pair. This is a 2D `numpy` array as shown below.

```
Q = np.zeros((env.n_states, env.n_actions))
```

- $N(s, a)$: the number of times the agent has tried each state action pair (s, a).
- α: the learning rate ($\alpha > 0$).
- E: the number of episodes in which the agent collect experience.

The Q-learning algorithm is in Algorithm 1. The pseudocode differs from that in the lecture notes because we are only executing the main loop for a fixed number of episodes, rather than checking for convergence of the Q-values. See section 2.3 for a description of the exploration strategies you may use. Note that Q-learning is *off-policy*; that is, we don't use the current policy to estimate the actual Q-value, assuming instead that a greedy policy is followed.

The SARSA algorithm is given in Algorithm 2. SARSA stands for state-action-reward-state-action. Like Q-learning, it applies a temporal difference update rule to learn the Q-values. Unlike Q-learning, SARSA is *on-policy* algorithm, meaning that the Q-values are estimated by using the agent's current policy.

2.3 Managing the Trade-off between Exploration and Exploitation

During active reinforcement learning, there is a trade-off between exploration and exploitation. You will implement three different strategies for managing this trade-off.

Algorithm 1 Active Q-learning

```
Require: $\alpha > 0$                                                    ▷ Learning rate
Require: $\gamma \in (0, 1]$                                             ▷ Discount factor
  $R(s) \leftarrow 0$                                                    ▷ Initialize reward function
  $Q(s,a) \leftarrow 0$                                                  ▷ Initialize Q-values
  $N(s,a) \leftarrow 0$                  ▷ Initialize state-action visit counts (for action selection)
  for $i \in \{1, 2, \ldots, E\}$ do                                     ▷ Conduct $E$ episodes
    $s \leftarrow$ env.reset()                                           ▷ Get initial state
    while $s$ is not terminal do                      ▷ Take actions until the state is terminal.
      $a \leftarrow$ select_action($s$)               ▷ Select action according to action strategy
      $s', r \leftarrow$ env.step($a$)        ▷ Take action $a$ and generate an experience $\langle s, r, a, s' \rangle$
      $R(s) \leftarrow r$                                                ▷ Update reward function
      $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(R(s) + \gamma \max_{a'} Q(s', a') - Q(s,a)\Big)$     ▷ Update $Q(s,a)$.
      $N(s,a) \leftarrow N(s,a) + 1$                                     ▷ Update state-action visit count
      $s \leftarrow s'$                                                  ▷ Update current state
    end while
  end for
  return Q
```

Algorithm 2 SARSA

```
Require: $\alpha > 0$                                                    ▷ Learning rate
Require: $\gamma \in (0, 1]$                                             ▷ Discount factor
  $R(s) \leftarrow 0$                                                    ▷ Initialize reward function
  $Q(s,a) \leftarrow 0$                                                  ▷ Initialize Q-values
  $N(s,a) \leftarrow 0$                  ▷ Initialize state-action visit counts (for action selection)
  for $i \in \{1, 2, \ldots, E\}$ do                                     ▷ Conduct $E$ episodes
    $s \leftarrow$ env.reset()                                           ▷ Get initial state
    $a \leftarrow$ select_action($s$)         ▷ Select initial action according to action strategy
    while $s$ is not terminal do                      ▷ Take actions until the state is terminal.
      $s', r \leftarrow$ env.step($a$)        ▷ Take action $a$ and generate an experience $\langle s, r, a, s' \rangle$
      $R(s) \leftarrow r$                                                ▷ Update reward function
      $a' \leftarrow$ select_action($s'$)        ▷ Select next action according to action strategy
      $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(R(s) + \gamma Q(s', a') - Q(s,a)\Big)$     ▷ Update $Q(s,a)$.
      $N(s,a) \leftarrow N(s,a) + 1$                                     ▷ Update state-action visit count
      $s \leftarrow s'$                                                  ▷ Update current state
      $a \leftarrow a'$                                                  ▷ Update current action
    end while
  end for
  return Q
```

<u>IMPORTANT:</u> To implement these strategies, you will need to select random samples and

apply the `argmax` operation. To ensure reproducibility for testing purposes, please use the following functions provided in `utils.py`.

- `sample_integer_from_categorical_distribution(P)`: samples an integer from a categorical distribution with probability distribution function `P`, which is a 1D numpy array
- `argmax_with_random_tiebreaking(v)`: Takes the argmax of the 1D array `v`, breaking ties with uniformly random selection from the maximal elements

Below are brief descriptions of the three strategies you will implement in this assignment.

1. **ϵ-greedy strategy:** ϵ is the probability of taking a random action. The agent takes a random action with a probability of ϵ and takes the action with the largest Q value with a probability of $1 - \epsilon$.
2. **Softmax action selection:** The agent selects each action with a probability that is proportional to the Q value for the action. The probability of choosing action a in state s is given below.

$$\frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}} \tag{1}$$

   T is a parameter called the temperature. T affects the shape of the distribution. When T is large, the distribution is close to being uniform. As T decreases, the distribution becomes better at distinguishing actions with different Q values. As T approaches 0, the distribution approaches a point mass with a probability of 1 for the action with the largest Q value.
3. **Optimistic utility estimates:** Another strategy for managing the trade-off is to modify the utility estimates instead of modifying the action selection strategy. Then select the greedy action with respect to the utility estimates, Q^+. Until the state-action pair (s, a) has been visited N_e times, the utility estimate is the maximum possible reward obtainable in any state (hence optimistic). Afterwards, the utility estimate is the expected utility, $Q(s, a)$.

$$Q^+(s,a) \leftarrow R(s) + \gamma \max_a f(Q^+(s,a), N(s,a))$$

$$f(u,n) = \begin{cases} R^+, \text{ if } n < N_e \\ u, \text{otherwise} \end{cases}$$