ASSIGNMENT #3 – COMP 3106 ARTIFICIAL INTELLIGENCE

The assignment is an opportunity to demonstrate your knowledge on reinforcement learning and practice applying it to a problem.

The assignment may be completed individually, or it may be completed in small groups of two or three students. The expectations will not depend on group size (i.e. same expectations for all group sizes).

Assignment due date: 2025-11-13

Assignments are to be submitted electronically through Brightspace. It is your responsibility to ensure that your assignment is submitted properly. Copying of assignments is NOT allowed. Discussion of assignment work with others is acceptable but each individual or small group are expected to do the work themselves.

Components

The assignment should contain two components: an implementation and answers to the questions below.

Implementation

Programming language: Python 3

You may use the Python Standard Library (https://docs.python.org/3/library/). You may also use the NumPy, Pandas, and SciPy packages (and any packages they directly depend on). Use of any additional packages requires approval of the instructor.

You must implement your code yourself. Do not copy-and-paste code from other sources, but you may use any pseudo-code we wrote in class as a basis for your implementation. Your implementation must follow the outlined specifications. Implementations which do not follow the specifications may receive a grade of zero. Please make sure your code is readable, as it will also be assessed for correctness. You do not need to prove correctness of your implementation.

You may be provided with a set of examples to test your implementation. Note that the provided examples do not necessarily represent a complete set of test cases. Your implementation may be evaluated on a different set of test cases.

The implementation will be graded both on content and use of good programming practices.

Submit the implementation as a single PY file.

Questions

You must answer all questions posed in the section below. Ensure you answers are clear and concise.

If the assignment was completed in a small group of students, you must also include a statement of contributions. This statement should identify: (1) whether each group member made significant contribution, (2) whether each group member made an approximately equal contribution, and (3) exactly which aspects of the assignment each group member contributed to.

Submit your answers to the questions as a single PDF file.

Implementation

Consider a two-player 13-coin game. In this game, the two players alternative taking one, two, or three coins from a bag. The player who takes the last coin from the bag loses. In this assignment, we will use temporal difference Q-learning to learn the optimal policy to achieve as much reward as possible in this game.

Assume that this is a fully observable environment: the players always know how many coins each player has, and how many coins are remaining in the bag. Assume that this is a discrete time environment, and at each time, your agent may take one, two, or three coins from the bag. Your agent and the opponent alternate taking turns.

The state of the environment comprises: how many coins are left in the bag, how many coins your agent has, how many coins the opponent has, and who the winner is (if it is a terminal state). We use the following string representation for environment states:

$$c_{bag}/c_{agent}/c_{opponent}/winner$$

Where c_{bag} indicates how many coins are left in the bag.
Where c_{agent} indicates how many coins your agent has.
Where $c_{opponent}$ indicates how many coins your opponent has.
Where $winner$ indicates who is the winner (A indicates your agent is the winner, O indicates the opponent is the winner, and $-$ indicates a non-terminal state).

As an example, the representation $10/1/2/-$ indicates there are 13 coins left in the bag, your agent has 1 coin, the opponent has 2 coins, and this is a non-terminal state. Alternatively, the representation $0/6/7/A$ indicates there are zero coins left in the bag, your agent has 6 coins, the opponent has 7 coins, and your agent was the winner (i.e. the opponent took the last coin).

We will represent actions in the following way. Note that not all these actions are always available to your agent (e.g. your agent cannot take 3 coins if there are only 2 coins left in the bag).
1: take one coin
2: take two coins
3: take three coins

In a winning terminal state, your agent receives reward equal to the number of coins it has collected. In a losing terminal state, your agent receives reward equal to the negative of the number of coins it has collected. There is zero reward associated with non-terminal states. That is, the reward r associated with a state s is:

$$r(s) = \begin{cases} c_{agent} \; if \; winner == A \\ -c_{agent} \; if \; winner == O \\ 0 \; otherwise \end{cases}$$

Use the following parameters:
Gamma = 0.90 (discount factor)
Alpha = 0.10 (learning rate)

Initially estimate the Q-function as:

$$Q(s,a) \; = \; r(s)$$

A few important notes for your implementation:
1. Use all the provided trials (which were generated under a fixed random policy) to learn the Q-function.
2. Iterate over all the trials multiple times until convergence is reached.

Your implementation must contain a file named “assignment3.py” with a class named “td_qlearning”.

The “td_qlearning” class should have three member functions: “__init__”, “qvalue”, and “policy”.

The function “__init__” is a constructor that should take one input argument (in addition to “self”). The input argument is the full path to a directory containing CSV files with trials through the state space. Each CSV file will contain two columns, the first with a string representation of the state and the second with an integer representation of the action taken in that state. You may assume only valid actions are taken in each state in the trial. Note that only states where it is your agent’s turn and terminal states are included; states where it is your opponent’s turn are omitted.

The function “qvalue” should take two input arguments (in addition to “self”). The first input argument is a string representation of a state. The second input argument is the integer representation of an action. The function should return the Q-value associated with that state-action pair (according to the Q-function learned from the trials in the directory passed to the __init__ function).

The function “policy” should take one input argument (in addition to “self”). The input argument is a string representation of a state. The function should return an integer representation of the optimal action (according to the Q-function learned from the trials in the directory passed to the __init__ function). In the case of a tie (i.e. multiple actions are equally optimal), the function may return any one of the equally optimal actions.

The “qvalue” and “policy” functions will be called after the constructor is called. They may be called multiple times and in any order. I recommend computing the Q-function within the “__init__” function (store it in a member variable) and implement the “qvalue” and “policy” functions as getters.

Attached are example inputs and corresponding example outputs. Note that your functions should not write anything to file. These examples are provided in separate files for convenience.

Example trial CSV file:
```
13/0/0/-,1
11/1/1/-,2
8/3/2/-,2
3/5/5/-,2
0/7/6/A,-
```

Example inputs to “qvalue” function:
```
8/3/2/-
2
```

Example output from “qvalue” function:
```
5.67
```

Example input to “policy” function:
```
11/1/1/-
```

Example output from “policy” function:
```
2
```

Attached is skeleton code indicating the format your implementation should take.

Grading

The implementation will be worth 70 marks.

50 marks will be allocated to correctness on a series of test cases, with consideration to both the Q-function and the policy. These test cases will be run automatically by calling your implementation from another Python script. To facilitate this, please ensure your implementation adheres exactly to the specifications.

20 marks will be allocated to human-based review of code.

Questions

Please answer the following questions. Explain why your answers are correct.

1. What type of agent have you implemented (simple reflex agent, model-based reflex agent, goal-based agent, or utility-based agent)? [3 marks]
2. Is the task environment: [7 marks]
    a. Fully or partially observable?
    b. Single or multiple agent?
    c. Deterministic or stochastic?
    d. Episodic or sequential?
    e. Static or dynamic?
    f. Discrete or continuous?
    g. Known or unknown?
3. For some cases (even with many trials through state space), the optimal Q-value for a particular state-action pair will not be found. Explain why this might be the case. [4 marks]
4. In the test cases provided, the trials through state space were simulated using a random policy. Describe a different strategy to simulate trials and compare it to using a random policy. [4 marks]
5. Suppose we use a large state space approximation for learning Q-values for this task.
$$Q(s,a) = \Theta_1 f_1(s,a) + \cdots + \Theta_n f_n(s,a)$$
Suggest a set of basis functions (or features) that could be used for the large state space approximation and explain why they are useful features. [4 marks]
6. An alternative approach to find the optimal strategy for this two-player adversarial game is to use minimax search. Suppose your opponent's reward function is analogous to your agent's.
$$r_{opponent}(s) = \begin{cases} c_{opponent} \; if \; winner == A \\ -c_{opponent} \; if \; winner == O \\ 0 \; otherwise \end{cases}$$
Describe what modifications (if any) you would have to make to use minimax search to find the optimal move for your agent to take in this environment. [4 marks]
7. Under what conditions will your methods using reinforcement learning yield better rewards than minimax search? Under what conditions will minimax search yield better rewards than reinforcement learning? Provide justification. [4 marks]

Grading

The questions will be worth 30 marks, allocated as described above.