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: 2024-11-14

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.

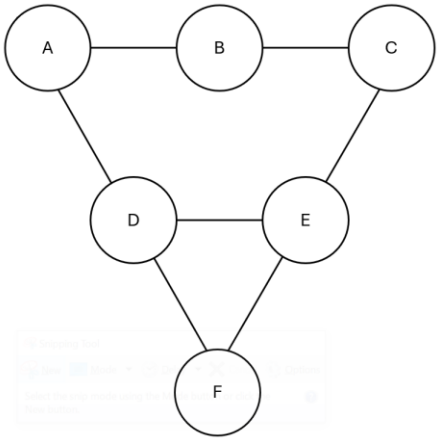
If the assignment was completed in a small group of students, you must also include a statement of contributions. This statement should identify: (1) whether each group member made significant contribution, (2) whether each group member made an approximately equal contribution, and (3) exactly which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

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

Implementation

Consider a cat and mouse game with six nodes, as illustrated below. In this game, the mouse tries to get to a node with cheese without getting eaten by the cat. In this assignment, we will use temporal difference Q-learning to learn the optimal policy for the mouse to achieve as much reward as possible.

In this environment, we will assume there are six nodes. The node in the top-middle (Node B) contains cheese.



Assume that this is a fully observable environment: the mouse always knows which node it is in and which node the cat is in. Assume that this is a discrete time environment, and at each time, the mouse may take one action to move to an adjacent node (which nodes are adjacent is indicated in the diagram above). Furthermore, at each time, with some probability, the cat may move to an adjacent node.

The state of the environment comprises which node the mouse is in and which node the cat is in. We use the following string representation for environment states:

$$P_{Mouse}P_{Cat}$$

Where P_{Mouse} indicates which node the mouse is in.
Where P_{Cat} indicates which node the cat is in.

As an example, the representation FB represents the mouse is in node F and the cat is in node B.

We will represent actions in the following way. Note that not all these actions are always available to the mouse (e.g. the mouse cannot move to node A if the mouse is in node E).
"N": do not move
"A": move to node A
"B": move to node B
"C": move to node C
"D": move to node D
"E": move to node E
"F": move to node F

The mouse receives large positive reward and the trial ends if the mouse reaches a node with cheese (even if the cat is in the same square). The mouse receives large negative reward and the trial ends if the mouse is in the same node as the cat and the node does not have cheese. The mouse receives small negative reward for all other states. The reward r associated with a state s is:

$$r(s) = \begin{cases} +10 \; if \; P_{Mouse} == B \\ -10 \; if \; P_{Mouse} == P_{Cat} \; and \; P_{Mouse} != B \\ -1 \; otherwise \end{cases}$$

Use the following parameters:
Gamma = 0.95 (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 a string representation of the action taken in that state. The i^{th} row of the CSV file indicates the state-action pair at time i. You may assume only valid actions are taken in each state in the trial.

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 string 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 a string 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:
FB,D
DA,E
EB,D
DB,N
DA,N
DA,E
EA,C
CB,B
BC,-

Example input to “qvalue” function:
DA
E

Example output from “qvalue” function:
2.55

Example input to “policy” function:
EB

Example output from “policy” function:
D

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. Suppose there is a state-action pair that is never encountered during a trial through state space. What should the Q-value be for this state-action pair? [2 marks]
4. For some cases (even with a long trial through state space), the optimal Q-value for a particular state-action pair will not be found. Explain why this might be the case. [3 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. Using the large state space approximation from above, the optimal policy is given below.
$$\pi(s) = \max_{a}(\Theta_1 f_1(s,a) + \cdots + \Theta_n f_n(s,a))$$
Consider learning the parameters $\Theta_1, \dots, \Theta_n$ to directly optimize the policy using the genetic algorithm. In this case, a candidate solution for the genetic algorithm is a set of values for the parameters $\Theta_1, \dots, \Theta_n$. For this case, suggest the fitness function, the genetic operator, and the mutation operator for the genetic algorithm. [6 marks]
7. 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. [5 marks]

Grading

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