HOME ASSIGNMENT 2, CMPE 260, SPRING 2023.
###
TOPIC: Sequential decision making with discrete state and action spaces.
###
LEARNING OBJECTIVES: to get hands-on experience with the theoretical concepts,
discussed in the class, such as MDP graph, Value/Policy Iteration, Q-learning,
SARSA, DQN (a bonus question). And, to get hands-on experience with efficient
implementation with broadcasting of multidimensional tensors. ###
###
###
This assignment is a part of the preparation for the midterm on the same topics.
You can work in teams of 2-3 students. Your can discuss your solutions with other
teams, but sharing your code or parts of it with other teams is plagiarism.
###
DUE: April 7, 11:59PM.

UTILITY FUNCTIONS / PSEUDO-CODE

5 possible actions: Up, Right, Left, Down, Stay
PSEUDOCODE: Actions = Dict(1=>(-1, 0), 2=>(0, +1), 3=>(0, -1), 4=>(+1, 0), 5=>(0, 0))
E.g., Right = (keep constant the first coordinate of state, increase the second coordinate of state by +1)

example for an obstacle (fence with length 3 blocks)
PSEUDOCODE: Obstacle = [(4, xObst) for xObst in 1:3]

PSEUDOCODE: function validState(s) = # returns true if (state, s, is within the maze boundaries) AND (s is NOT in the obstacles)

dS - size of the maze with dimensions: dS x dS //
dA - number of actions
Goal - goal state.
PSEUDOCODE: function BuildMaze(dS, dA, Goal)

    # dynamics tensor with dimensions: |dS| x |dS| x |dA| x |dS| x |dS| x 1, where the
    # dimensions are S_1, S_2, A, S_1', S_2'. e.g., S_2 is the current second coordinate of the state
    # and S_1' is the first coordinate of the state at the next time step.
    Ps′_sa = zeros(dS, dS, dA, dS, dS)

    # the reward tensor with the same dimension as the dynamics
    # reward is -1 on every state, and 0 at the Goal state.

```
    Rs′sa  = -ones(dS, dS, dA, dS, dS)

    # iterate over the valid states
    for s in filter(validState, (x->x.I).(CartesianIndices((dS, dS))))
        if s ∈ Goal
            Ps′_sa[s..., :, s...] .= 1.0 # all the actions get prob 1 at the goal
            Rs′sa[s..., :, s...]  .= 0.0 # all the actions get reward 0
            continue
        end

        for a in Actions # the same action set at each state
            # if "next state is valid" move to it, otherwise stay at place
            s′ = validState(s .+ a[2]) ? s .+ a[2] : s
            Ps′_sa[s..., a[1], s′...] = 1.0
        end
    end
    "sanity test:" forall a, s : sum_s′ Ps′_sa = 1
    return Ps′_sa, Rs′sa
end
```

TASKS

Task 1
Build your maze with dimensions 10x10 and 3 fences, and the goal state (exit)
in one of the corners of the maze. Visualize the maze layout on 2D plot.

The tasks 2 - 4 are for the model-based setting, where both p(s′ | s, a) and R(s′, s, a) are known.

Task 2
Implement the Policy Evaluation (PE) algorithm for a deterministic policy, π.
The dimensions of the policy π[a | s] are |dS| x |dS| x |dA| x 1 x 1 x 1.
There is a single possible action at every state.
E.g., π[UP | some_state] = [1, 0, 0, 0, 0], see 'Actions' above.
E.g., π[Stay | another_state] = [0, 0, 0, 0, 1], see 'Actions' above.

Evaluate a random deterministic policy, π. Plot Value of a random policy on 2D plot.

Instructions:
The policy is stationary, which means π[a′ | s′] is permute_dimensions(π[a | s], dim1->dim4, dim2->dim5, dim3->dim6)
Use broadcasting '.*', e.g.,
p(s′ a′ | s, a) = π[a′|s′] .* p(s′ | s, a)
sum_s′p(s′ | s, π[a|s]) .* V[s′], where V[s′] is the value of the next state with dimensions 1 x 1 x 1

x dS x dS x 1.
The value of the current state has dimensions dS x dS x 1 x 1 x 1 x 1.
"V of the next state" is permute_dimensions("V of the current state", dims1->dims4, dims2->dims5)

Task 3
Repeat Task 2 with manually setting the optimal actions in the radius of 2 states from the goal state.
Explain your observations.

Task 4
Implement the Policy Improvement (PI) Algorithm, and find the optimal policy π*.
Visualize the optimal value function, V_i, on a 2D plot at 3 different iterations, i, of PI.
Explain your observations.

The next tasks are for the model-free setting, where neither p(s′ | s, a) nor R(s′, s, a) are known.

Task 5
Write a function, s′, r = step(s, a), that receives the current state, s, and the current action, a, and
returns the next state, s′, and reward, r.

Generate 10 trajectories from 10 different initial states, using a random policy.
Generate 10 trajectories from 10 different initial states, using the optimal policy from Task 4 above.
Explain your observations.

Task 5
Implement Q-learning algorithm for the tabular case, where Q function is given by a table.
Plot an accumulated reward as a function of the iteration number of Q-learning algorithm
for 5 runs of Q-learning from scratch. Plot an average curve of the 4 runs of Q-learning,
and the variance (use fill_between). Explain your observations

Task 6
Repeat Task 5 with the SARSA Algorithm.
Explain your observations.

Task 7 (bonus 20 points)
Repeat Tasks 5 and 6 with the DQN algorithm, where Q is represented by a neural network with a few
feedforward layers. Compare the results with and without Experience Replay, and with and without a

separate network for the target. Explain your observations.

What to submit:

1. a runnable code, which enables to reproduce your results
2. a PDF file with all the plots, explanations, and the code.

Please submit two separate files, rather than a single ZIP, in Canvas.