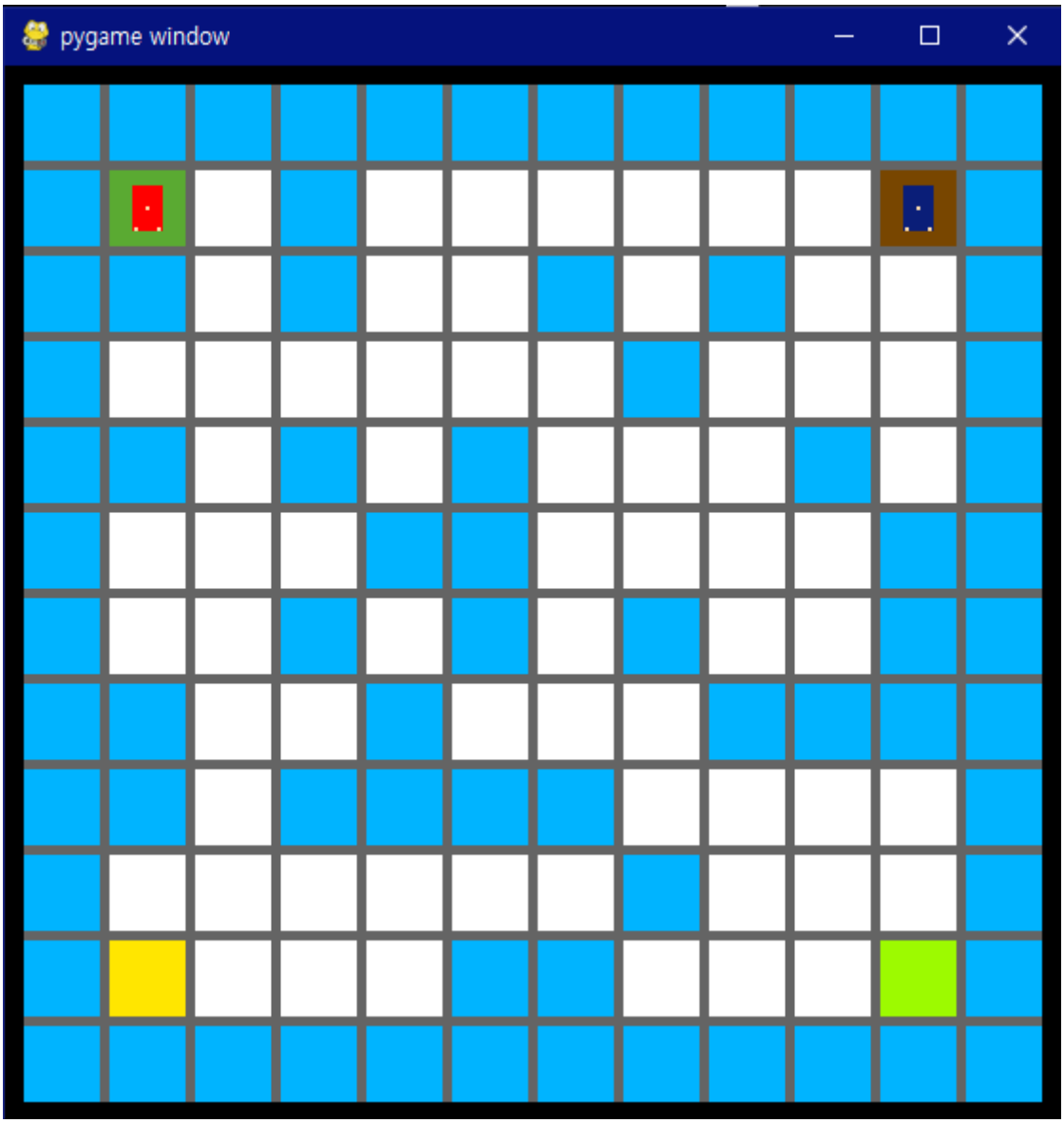
KAIST CS270 Intelligent robot design and programming
Assignment 4 – Deep Q-Learning in robot simulation (path planning)

CS270 Assignment 4 – Deep Q-Learning



Introduction

In this assignment, we will implement deep Q-learning algorithm with the simulator used in assignment 3. In the course material, deep Q-learning algorithm is introduced. This assignment is designed to show the difference between the original Q-learning algorithm and Q-learning supported with deep learning algorithms. Again, you want to train your robot with a simulator so that it can learn how to solve the frozen lake problem and avoid blue sections while managing to reach the destination point. There are “three file that will be provided to you: student.py, simulator_hidden.py, and simulator.py. simulator_hidden.py serves as the backbone of the simulator; this file creates the simulator and gets it running. student.py is where you will code your algorithm. Last but not least, simulator.py serves as a middleman between simulator_hidden.py and student.py; it gives access of some useful functions implemented in simulator_hidden.py to student.py.

Task

Your task for this assignment is to manipulate the robot to reach to the goal position. However, now the map is bigger ((8, 8) → (12, 12), there are more obstacles (8 → 30) and map will constantly change for each episode just like assignment 3. Last but not least, you have to manipulate two robots to reach their respective goals. Starting point ((1, 1) for the red robot, (10, 1) for the blue robot), goal point ((10, 10) for the red robot, (1, 10) for the blue robot) and starting orientation (both ‘S’) will be fixed for every episode.

In assignment 3, we had a single terminating condition: if the robot reached the goal or it steps into the blue section on the map. Now, this changes to whether both robots are finished (by either reaching the goal or stepping on to the lake). Furthermore, we have one more terminating condition: two robots cannot be at the same section on the map concurrently as applying more pressure on the frozen lake will break the ice.

Preparation

(1) Pytorch

As you've experienced in assignment 1, you are required to design your own neural network in this assignment that will guide your robot to the final destination. If you forget how to set up pytorch, please refer to the doc file of assignment1 or lab session 1.

(2) Simulator

As we now have two robots, the simulator has been modified little bit for you to manipulate two robots. Following are changed functionalities.

1) move_forward(), turn_left() and turn_right() now accept an integer as the input. If the input is 1, the red robot moves forward or turns to either side. If the input is 2, the action will be executed by the blue robot.
2) Previously, three functions above returned the state of the robot after the action is executed. Now, those functions, regardless of the input, return the state of two robots simultaneously. To be specific, current state returned by those functions is a 10-dimensional tuple consisting of the followings:
   A. Coordinate of the red robot
   B. Orientation of the red robot
   C. Sensor values obtained by the red robot (2 if there's the other robot, 1 if there's a lake, 0 otherwise)
   D. Red robot's status (True if the red robot is either wasted or reached its goal point, False otherwise)
   E. Whether the red robot reached the goal (True or False)
   F. Coordinate of the blue robot
   G. Orientation of the blue robot
   H. Sensor values obtained by the blue robot (2 if there's the other robot, 1 if there's a lake, 0 otherwise)
   I. Blue robot's status (True if the blue robot is either wasted or reached its goal point, False otherwise)
   J. Whether the blue robot reached the goal (True or False)

   If not specifically mentioned above, the dimension or the return value of each property is same as was used in the assignment 3.
3) Robot (either red or blue) stops moving or rotating regardless of functions used to manipulate them if they reach their respective goals (specifically, if E or J property from above returns True).
4) As now we have a 12 x 12 map, the simulation might be too large if you're working on devices with small monitors. If that's the case, modify the value of variable named sensor in simulator_hidden.py (line 30) from 2 to 1.

Assignment 4 Implementation:

You are required to fill in the blank areas in student.py. Do not touch the other two files, and do not import simulator_hidden in student.py. Implement DQN with the following structures.

Tasks: You are given 6 tasks in this homework.

(1) Task 1 – design your own model

Subtasks	To do
(1) Design state and action	Your model will take the current state as the input. There are different values that can be utilized as the current state, such as current position, sensor outputs from three different sensors, and current robot orientation. You can use all the information provided from the robot, or select some of them as the input. For the simplicity of the assignment, you are only allowed to use move_forward, turn_left and turn_right to manipulate each robot. Unlike the previous assignment, **you are not allowed to define your own actions** by combining existing actions. However, if needed, you can include one custom action that does not apply any positional or rotational change to the robot.
(2) Decide input shape and output shape	You need to make sure that the model input shape does fit with the input shape of the first layer. In addition, shape of the output tensor that your model should generate should match with the number of total actions possible.
(3) Design you own model	You have learned different layers that can be used as the inner structure of you model. Generate you own model so that it can substitute Q-table well.

(2) Task 2 - select action

Subtasks	To do
(1) Create current state and put it to select_action function	Combine information that you wish to consider as a current state to generate a tensor (pytorch tensor) that can be used as the input of your model (torch.from_numpy(), np.concatenate(), functions might be useful when you generate a tensor)
(2) Select actions using the model	Use you model to generate action. Select_action() function should return the index of an action. The function torch.argmax() might be needed to pick maximum index from model output.
(3) Exploration	While training the model, the model should be able to explore different search space, which means that it should do random actions with certain probability. Design select_action function so that it randomly not follow the model but generate a random action.

(3) Task 3 – replay memory

Subtasks	To do
(1) Push transition to the memory	Given a single transition that consists of a current state, an action, a reward and a new state, push function store the transition into its memory. The memory length should not exceed capacity that are pre-determined when the memory is declared (self.capacity).
(2) Sample transitions from the memory	Given a batch_size, you need to sample a random set of transitions from memory. Sampled set should have a size of the batch size.

(4) Task 4 – optimize model

Subtasks	To do
(1) Generate state_action_values	Use pre-built memory, sample certain batched transitions. Then generate state_action_values from policy net. The policy net is the network that is used to generate transitions.
(2) Generate state_values for the new state	Generate state values for the new state using target network. New state values are the maximum value among the state action values generated by the target network.
(3) Generate expected state action values	Calculate expected state action values using the equation in the lecture note with reward and new state value function.
(4) Set loss and train the network	Check sudo code in the lecture note to set an appropriate loss function and train policy net.
(5) Update target network	Choose hyperparamter C, and update the parameter of the target network as that of policy network for every C step (load_state_dict() function might be useful)

(5) Task 5 – save model and generate test code

Subtasks	To do
(1) Save the parameter of final policy net and finish test_network function	Finish test_network function so that it will test trained network.

(6) Task 6 (optional) – design your own reward function

After implementation, you can run the code to see if you did a good job:

a. Run the code and watch the robot as it trains (or not if you disabled animations)
b. Watch during testing mode to see whether the robot succeeded the test map or not. Obviously, just because the robot succeeded in the test map doesn't mean it will succeed in all cases.

Grading

Several fixed test maps will be used for grading. These test maps will not be provided. Grading will take into consideration the following:

1. The number of test maps succeeded (including number of the robots that reached the goal)
2. The number of steps it takes for each success

Submission Guidelines

Please submit a zip file named "studentid_studentname.zip" containing the logs folder, the saved policy network, and student.py to KLMS by June 17th, 11:59 pm.

We will use "test_network" function (task 5) to test your assignment, so please make sure that the test_network function is loading policy_net correctly and run testing.

Additional help

To make implementation easier, we also include student_prev.py which includes one implementation of DQN when it is applied to setting that is same with assignment 3 (same map size, same number of obstacles, one robot, etc). You can refer to this file if you get stuck.

Implementation in this file performs quite nicely when applied to setting of assignment 3. However, now we have more things to consider and dimensions of input, output, actions and etc might be larger than how they were in assignment 3. So, it might be necessary to consider different hyper-parameters.

We recommend you to run this file to get a grip of how robots are trained using DQN. To run this file, change its name to 'student.py' and run it with simulator.py and simulator_hidden.py from assignment 3, not the ones we provide in assignment 4.