ELEC-E7851 User Interfaces, 2020

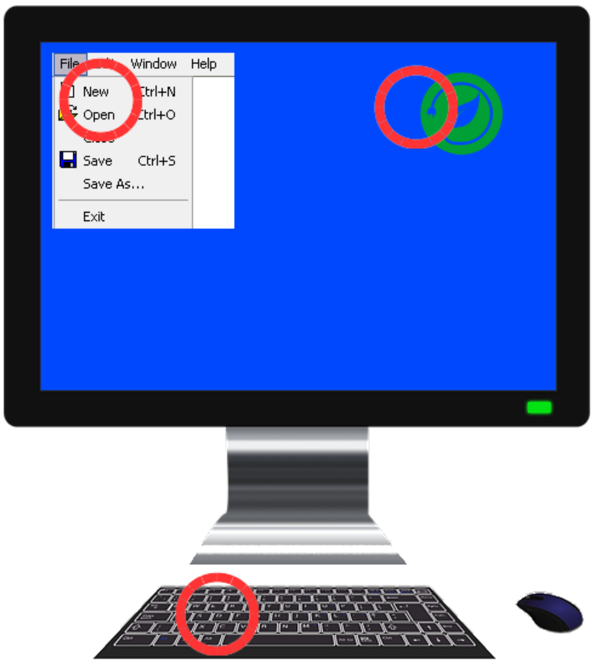
Aalto University

Antti Oulasvirta (Tasks by Jussi Jokinen)

Assignment 5: Reinforcement Learning I

General: There are two tasks in this assignment, each worth max 5 points. We believe 5.1. will be easier, so we recommend that one to start with. Both tasks utilize Python code, provided with this assignment. Please note that this is not a notebook file, but an ordinary Python code file. You can run it in any Python3 environment, as long as the standard libraries are in order.

Submission: Please submit one pdf file, which contains all assignments that you chose to do (so, either 5.1. or 5.2. or both). It is recommended that you first do 5.1. and only then consider 5.2.



A5.1. Learning transition probabilities (5 p)

Background: The end of the lecture introduced a simple decision process, where the user (or the agent) needs to solve the problem of opening a software (e.g., an email viewer) on a desktop computer. The user has different actions available, such as moving the mouse and clicking it, and the environment (the UI) can be in different states (e.g., mouse cursor can be in different places). The transition function of the environment defines how the user's actions change the states of the environment, depending on the current state.

All transition probabilities in the example used in the lecture were implicated to be 1.0, meaning that the same action always lead to the same next state, given the current state (e.g., going "right" in "s1" always resulted in "s2").

Goals: Your task is to consider how changing the transition probabilities changes the Q-values and therefore the optimal behaviour policy of the agent.

Materials: The code is provided and documented as *mdp-computer.py*. Please read through the code and make sure you understand its logic. The lecture went through the logic (*mdp.py*, which simulated the "mouse in the maze" scenario, is provided for reference, but is not needed for the assignment). You can refresh your memory and read the documentation to get a more thorough understanding. Notice that the transition function in the code is just a list of if-then rules. Of course, in a more complicated task, this transition function would be a more complicated program describing the dynamics of moving and clicking the mouse.

Subgoals:

1. **Familiarize yourself with the model**. Without changing any code, run the Python code to train the model. Report the optimal path that the agent would take as state-action transitions from the start to the finish. Also report the Q-values that the agent used to choose its actions. For this exercise, set the epsilon value to 0.0 *after* teaching the model, in order to get the optimal behavior without potential exploration.

2. **Add the "press key" action and its transition logic.** The lecture mentioned that pressing a key might be the fastest way for an expert agent to achieve its task. Implement a "press key" action, such that taking this action has a reward of -2, and it transitions, with 100% probability, the task to being finished (so, same as e.g., clicking the mouse when it is on the icon).

   You should report the Q-value of this action in all possible states, and indicate all states where it is optimal to press the key. *Note that the more states the agent has to explore, the longer it takes for the model to converge. Make sure you set the number of training iterations fairly large. Training the model might take some seconds even in these simple environments.*

3. **Change the transition probability of the "press key" action.** By making a *random.random()* call in the update_environment() function when the agent takes the action "press key", vary the probability of the key press resulting in finished state. Try probabilities 0.2, 0.6, and 0.9: if the key press fails, the reward is the -2, but the state is not set to finished and instead remains on the current state where the key was pressed.

   Please report the Q-value of pressing the key in the first state after *reset()*, given these probabilities. Please discuss in how these Q-values reflect the implemented dynamics of the environment. *Again, note that you might need to run the model for a many iterations to get converged Q-values. You can print the relevant Q-value as the model progresses and see when it starts to stabilize. The final Q-values will also slightly oscillate depending on the random outcome of the last key presses. You can report a mean of multiple runs to get a more reliable estimate.*

A5.2. Define an interaction task as MDP and simulate different users conducting it

Background: While any sequential interactive task can in principle be defined as an MDP, the problem often is in coming up with a neat and realistic description. This task lets you have a say in the kind of environment you wish to build to simulate how users recover from an error, given their experience with it.

Goals: Your task is to define all the elements of an MDP: the state and action spaces, the transition function, and the reward function, for the described interactive task.

Materials: The task under investigation simulates error recovery: the user is conducting a task with a goal in mind, but by a random chance the system creates an error, prompting the user to solve it before the task can continue. For simplicity, you can consider this as a "maze", where states are rooms in the maze, and actions (the four cardinal directions) transition the agent from one room to another. However, you can also implement the transition function as your own "mini-program", which has an interface (actions that can be taken and a way to observe its current state). The "mini-program" will be more fun to do, but it also OK to complete the task using the maze metaphor.

The task of the user is to get from the starting state (e.g., "s1") to the final state (e.g., "s4") trough at least two intermediate states (e.g., "s2" and "s3"; e.g., by taking the action "right" in each step, wherein "left" takes to the previous step, and "up" and "down" do nothing in the task proper).

The error is triggered on an intermediate state, e.g., "s2", and it is a complicated subtask in itself, of at least 6 connected states (e.g., "e1"–"e6"), where there is only one way out (and multiple incorrect paths; you must specify the error subtask such that only a correct sequence of actions result solving the error), e.g., "e6" leading back to "s3", such that the task proper can be finished. Note that there is no path directly back from the start of the error ("e1") to the task proper ("s2") – this transition is one-directional.

The task

Formalise the described problem solving task: your response should define all of the elements of a MDP. Your answer should specify S, A, T, R (set gamma to 0.9).

Train two agents. The "novice" agent never gets to see the error, that is, the probability that action "right" in "s2" results in "s3" is 1.0. Then, after the novice agent is trained to do the task proper (again, use very many iterations to guarantee convergence), change this probability to 1.0, i.e., force the error to trigger.

The "expert" agent is trained with the error probability as 0.5. After enough iterations, the expert solve both the task proper, and recover efficiently from the error state.

Run both the novice and the expert model through the (forced) error multiple times. Report how much faster the expert resolves the error compared to the novice. Report on the gradual learning of the novice into an expert, as it learns to solve the error state and transitions into an expert (e.g., visualize the number of steps used to reach the terminal state on y-axis, and number of training episodes on x-axis of a scatterplot).