Introduction to Machine Learning

Reinforcement Learning Exercise

These exercises should be solved using python notebooks (Jupyter) due to the ability to generate a report integrated with the code. It is assumed you are proficient with programming. All answers must be justified and the results discussed and compared to the appropriate baselines.

Imagine a situation where a robot (smiley) needs to **learn the sequence of actions** that takes it from an initial position (state 1) to the energy plug (marked with X). Imagine that it can experiment learning a simulated (simplified) environment. Something like the scenario depicted in Fig. 1.

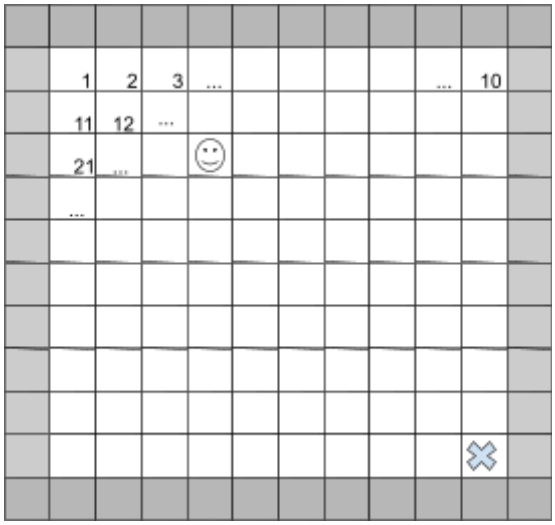

Fig. 1 The robot/agent starts in the top left corner and its goal is to reach the plug, in the bottom right corner. Grey squares represent walls. Some state numbers are depicted as example.

Suppose (by oversimplification) that the room can be divided in squares and these "areas" (that we will call states) are numbered so that the robot can identify each different position in the room. Imagine also that the robot has 4 actions (up / down / left / right) and these go from the middle of one square to the middle of an adjoining square. From this robot's point of view it knows it is in state 1 and if it moves left it will receive information that it arrived at state 2, if it moves down it will be informed of arriving at state 11, if it tries to move in another direction it will be informed that it remained in state 1.

Upon arrival at state 100 it will receive a reward.

1. Build the simulation environment:

    a) Build the state-transition function (*s' = f(s, a)*), where a state *(s)* and an action *(a)* are given as argument, and a new state (the arrival state, *s'*) is returned, so that: f(1, left) = 2, f(1, down) = 11, f(1, up) = 1, etc.

    b) Create a reward function *r(s)* that will reward all states with 0 and the goal-state (state 100) with 100 points.

    c) Program a function that will randomly choose an action.

    d) Define the end of the episode: When the robot reaches the goal-state (the plug, marked with an X, state 100) it should be returned to the initial position after getting the reward.

    e) Execute this function for 1000 steps and repeat 30 times. Measure the average reward per step in these 1000 steps. Calculate average and standard-deviation of number of steps to reach-goal, run-times and rewards for the 30 tests. These will be the baseline results and they will be used to test if the system is doing better than just random guessing in the future.

    f) Represent the average and standard-deviation (reward, steps and run-times), each in a different box-plot with vertical boxes.

Tip on results presentation for all exercises: In all situations that have any stochastic process (randomness or pseudo-randomness) involved, one needs to 1) store the random seed to repeat the same exact experiment if necessary, 2) repeat the experiment 30 times with the same parameters, and 3) calculate average and standard deviation of the 30 tests for each measured quantity. A common graphical representation of these is the box-plot with whiskers.

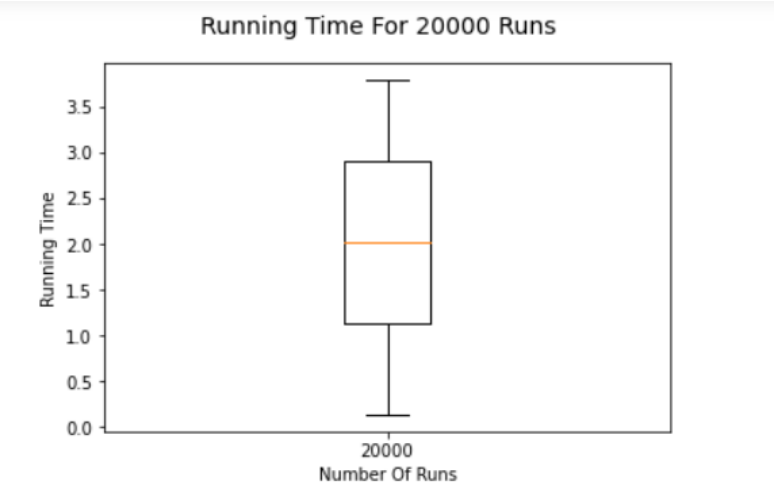

Fig. 2 Box-plot example

2. Create a matrix Q, indexed by the state index, and the action index Q(s, a) and make sure it is initialized with zeros. When arriving at a state *s'* **update the utility of the state where the robot came from *(s)***, using the following update-function:

$$Q(s, a) = (1 - alfa)\, Q(s, a) + alfa * (r(s') + discount * (max_{a'} Q(s',a')))$$

where $max_{a'} Q(s',a')$ is the best $Q(s',a')$ for all actions a' available at state s' and $r(s')$ is the reward given at state s'. Use the following values for *alfa* and *discount*: *alfa = 0.7* and *discount = 0.99.*

Can you now tell which is the best sequence of actions using **the information** on *Q*? And the best action from any given state?

a) Do a random walk (like in exercise 1) and execute this update-function after each state transition, for 20000 steps in each experiment and repeat the experiment 30 times. In each of the 30 experiments, at steps 100, 200, 500, 600, 700, 800, 900, 1000, 2500, 5000, 7500, 10000, 12500, 15000, 17500, 20000 (or other intermediate points that are deemed useful) stop to run a test.

A test consists of running the system for 1000 steps using the current Q table (without changing it) and always choosing the best action at each step. Measure the average reward per step in these 1000 steps.

Measure also the runtime of each full test (all 20000 steps) and calculate average and standard-deviation of run-times for the tests. Plot the steps (x-axis) vs avg reward (y-axis) of the tests at the measured points. A series of boxplots can also be used for a more informative view of the evolution of the robot's behavior.

Depict the final Q-table using a heatmap.

Tip: A heatmap is a good way to visualize the information in matrix Q. If you need to see the full policy, to represent the Q table, the best process is to have a heatmap of the maximum quality for each state and / or a matrix with the best action for each state.

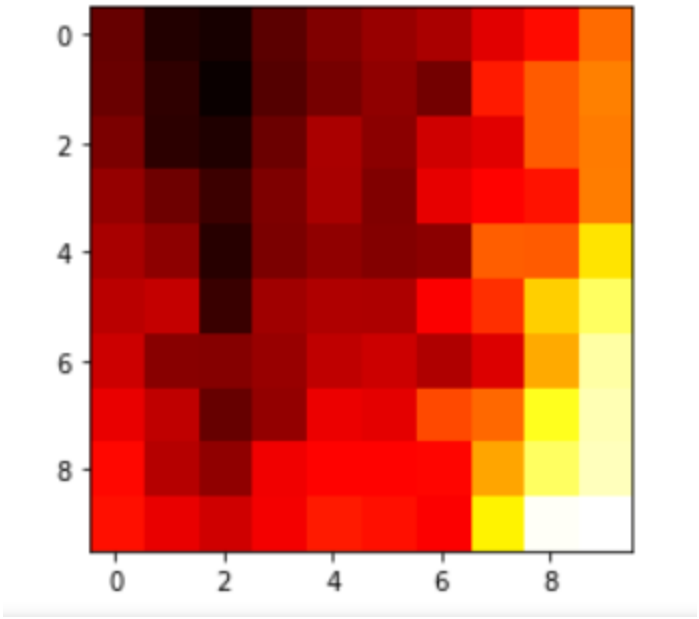

Fig. 3 Heatmap example (depicting maximum quality per state, from black (0) to white(100))

b) Do the same test as the previous example, but instead of random walk use always the Q-table values to choose the best action. Be careful to break ties randomly.

Compare tests a) and b) and draw your conclusions.

3. Use a mix of both strategies outlined above: Include a term *(greed)* in the action selection function that will determine the probability of choosing a random action. For example if *greed* is 0.9, approximately 10% of the actions chosen should be random, the remaining 90% should be the best action available according to Q. If greed is low, for example 0.2, approximately 80% of the actions are random. Try 3 different greed parameters and compare the results. Finally, try an increasing *greed* parameter starting around 30%, for the first 30% of the test steps, and slowly increasing until 100% by the end of the test. Compare test results and the Q tables.

4. Change the simulation to include walls (as in Fig 2) and that bouncing off a wall gives a small penalty reward (-0.1). Compare with previous results.

Optional

5. Imagine that the same action does not always take the robot to the same state. With a 5% probability it can take the robot to any neighbouring state of the current state. How does that affect the result?

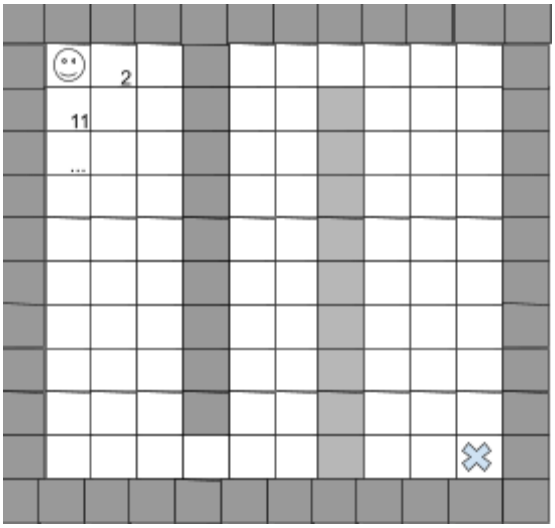

Fig. 4 Walls make the problem harder. Use penalty rewards (small compared to the final reward) to keep the agent in the right track.

6. Imagine now a situation, closer to the real scenario, where states are not numbered and the agent can only perceive its position by the echos on the walls. The agents' "perception" of what is around it is an array of floating point values that represent the distance to the wall for each side UP, LEFT, DOWN, RIGHT, e.g. NA, 0.56, NA, 0.14, means: no walls found UP, wall at 0.56 meters to the LEFT, no walls DOWN, wall 14 cm to the RIGHT. How can these states be simplified into a number that can be used as an index? what are the risks of this transformation in a scenario such as the one in Fig 4 (think about the states in column 2 and 8 for example)?