Machine Learning Summer School 2018 - Reinforcement Learning Workshop - Assignment 1

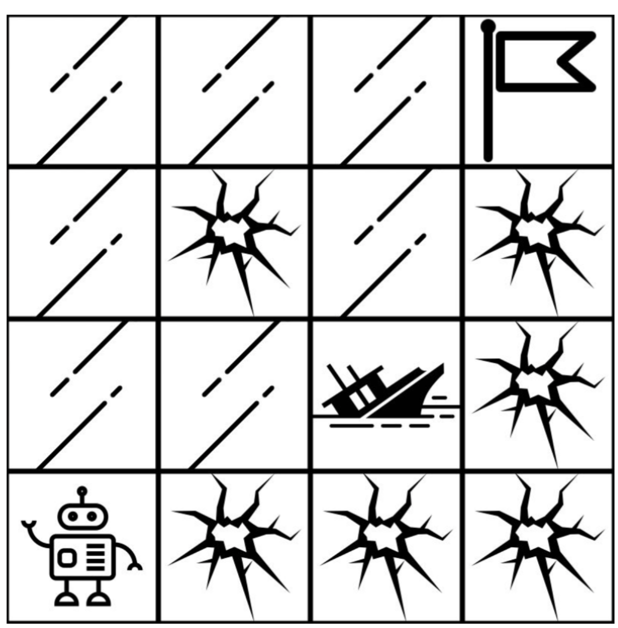
This document describes a simple (yet challenging) environment for which you will develop reinforcement learning agents. The first section presents the environment, the next one contains questions and suggestions about what to do next.

World Description

The ice world is a small, 4x4 icy grid. A robot is tasked to cross through the ice to reach the opposite side of the small world, while, if possible, collecting some treasure from a shipwreck embedded in the ice.

Unfortunately, the ice has been melting, is slippery, and is already cracked in several places. The robot must avoid falling through the cracks! Due to the slippery nature of the ice, the robot risks sliding on it at each step.

The world can be represented as follows:



The robot tries to take one step on the ice, one cell at a time, vertically or horizontally (not diagonally). At each time-step, the robot has a 5% chance of slipping on the ice, and go all the way to the side of the environment. For instance, if the robot tries to go up from any cell in the first column, it may slide across the environment and end up on the top-left cell (or in any crack it may encounter while sliding). The robot cannot move outside the environment, so trying to go up when in the first row has no effect.

Reaching the goal rewards the robot with 100 points. Passing on the shipwreck will allow the robot to collect some of the treasure inside: each time the robot passes there it will get an additional 20 points reward. If the robot falls through the cracks is destroyed (ending the episode), and so it's penalized with -10 points.

More formally, the environment has a total of 16 states (one for each possible position of the robot in the ice world). The robot has 4 actions available ('UP', 'RIGHT', 'DOWN' and 'LEFT'). Each action moves the robot in its direction with probability 0.95. When the robot tries to move outside of the grid, the action will have no effect with probability 1.

Slipping will happen with probability 0.05, and it results in the robot moving in the direction specified by the action up to either the grid border, or to the first crack (into it). (NB: it is possible for both slipping and not slipping as a result of an action to result in the same state, i.e., when the robot is one step away from the edge in the direction of movement, or when the agent is already at the edge and cannot move in that direction any further. Therefore, the transition function for states near the edge (for a given action/direction) looks different from states further away from the edge.)

Questions

The first two questions cover basic aspects of Reinforcement Learning, and should be implemented for you to be properly trained at RL. The other questions are optional, as they illustrate challenges and ways to improve RL algorithms.

Implementations of the environment described above are provided in Python. You can also reimplement the environment yourself in any language you want, but we strongly advise you to use our implementation.

1. Implement Value Iteration on the environment. Our implementation of the environment provides transition probabilities and rewards, so you can focus on the VI algorithm without having to fully understand what the environment does. Your implementation should output the optimal deterministic policy for this environment, along with the value of all 16 states.

2. Implement Q-Learning on the environment. Compare the values output by Value Iteration to your Q-Values. Does Q-Learning lead to the optimal policy? Does it find the same values as Value Iteration?

3. Optional: Implement Policy Iteration and compare the output and runtime of Value Iteration and Policy Iteration.

4. Optimal: Implement SARSA and compare the performance (i.e., cumulative reward) during learning of Q-learning and SARSA. Is there a difference, and if so, how can this be explained?

5. Optional: Extend your Q-Learning implementation with Experience Replay. Does it still find the optimal policy? How many episode does your agent need in order to learn (with and without ER)?

6. Optional: Extend Q-Learning with Eligibility Traces. Combining eligibility traces with experience replay is difficult (so don't do it at first). Why, and how would you do it?

7. Optional: Do you see any other way to increase the sample-efficiency of Q-Learning. Why is sample-efficiency so important in real-world applications of reinforcement learning ?