
Division of Engineering Science
UNIVERSITY OF TORONTO

ROB311: Artificial Intelligence

Project #4: Reinforcement Learning

Winter 2022

Overview

In this project, you will dabble with reinforcement learning algorithms. First, you will work with with two closely-related techniques for sequential decision making: *value iteration* and *policy iteration* for Markov decision problems. Thereafter, you will take a step back to understand the exploration-exploitation tradeoff by solving the *multi-armed bandit* problem. The goals are to:

- understand value iteration and the use of the Bellman update equations;
- compare value iteration with policy iteration for the same problem domains; and
- solve a classic reinforcement learning problem, referred to as the multi-arm bandit problem.

The project has three parts, worth a total of **50 points**. All submissions will be via Autolab; to prevent the usage of Autolab as a debugging tool, the maximum submissions are limited to **14** . To complete the project, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below. The due date for project submission is **Friday, April 14, 2022, by 23:55 EDT**.

Similar to Project #3, each part already has some basic code in place for you to start with. There are three types of files provided to you in the handout package: support files (named `mdp_*.py` or `mab_*.py`), template files (named `part*.py`) and test files (named `test*.py`). The students are only required to complete and submit the template files. To minimize the number of submissions, please use the test files to run preliminary tests on your function implementations before submitting on Autolab.

Part 1: Markov Decision Processes via Value Iteration

Value iteration is a well known method for solving Markov decision problems (processes). This iterative technique relies on what is known as the 'Bellman update' (see original work by Bellman in 1957), which you will code up as part of the project. Your tasks are to:

1. Write a short function, `get_transition_model()`, that generates the state transition model (matrix) for the simple cleaning robot problem described by Figure 1. This transition model will be needed to solve an instance of the cleaning robot MDP (see next bullet).
2. Implement the value iteration algorithm given in AIMA(4ed) on pg. 653, which accepts an MDP description as input (states, possible actions, transition matrix, rewards, discount factor) and produces an epsilon-optimal policy (and a utility function). The function, `value_iteration()`, will also accept a threshold (ϵ) and a maximum number of iterations to run.

You will submit the completed template files `part1_1.py` and `part1_2.py` through Autolab. We will test your code on several problems, including on the grid world example given in AIMA(4ed) on pg. 646! We have included an environment file (`mdp_grid_env.py`) in the handout package that defines the variables which are required for the AIMA problem. The function `init_stochstic_model()` (i.e., the function that

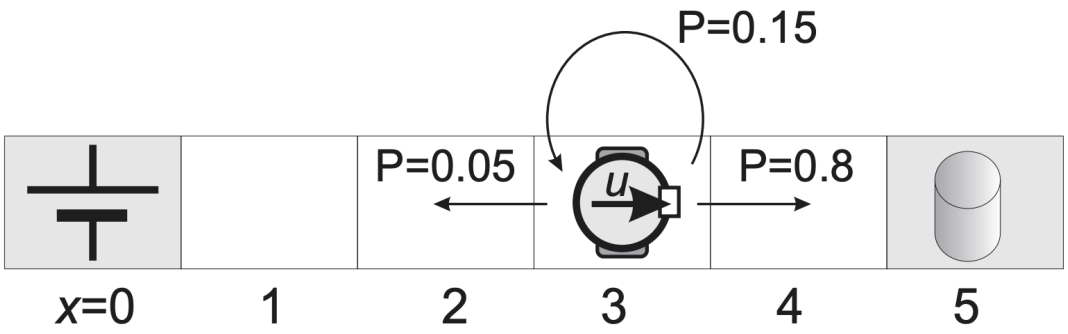

Figure 1: A simple cleaning robot problem. There are six states; the robot wants to put rubbish in the bin (State 5) and to recharge its batteries (State 0). Both of these states are terminal states (i.e., once reached, the robot does not leave the state). The robot may choose to move left or right to an adjacent cell. Due to environment uncertainties, such as a slippery floor, for example, state transitions are not deterministic: when trying to move in a certain direction, the robot succeeds with a probability of 0.8; with a probability of 0.15 it remains in the same state, and with a probability of 0.05 it may move in the opposite direction. The reward in State 0 is 1, in State 5 is 5, and is zero otherwise.

generates the transition model) has deliberately not been implemented, so as not to give away all the test cases. After implementing this function, you should be able to test your solution on more complex problem instances, as done in Autolab. Thus, you have the option to create additional tests for the grid environment using the provided test files—this should help with debugging, etc.

Part 2: Markov Decision Processes via Policy Iteration

As discussed in the lectures, value iteration is not the only way to solve an MDP; another popular alternative is policy iteration. The policy iteration framework is different than that of value iteration: we begin with an initial, sub-optimal policy (possibly random), and then refine that policy. Your task is to:

1. Implement the policy iteration algorithm given in AIMA(4ed) on pg. 657, which accepts an MDP description as input (states, possible actions, transition matrix, rewards, discount factor) and produces an optimal policy (and a utility function). The function, `policy_iteration()`, will also accept a variable that specifies the maximum number of iterations to run.

You will submit the completed template file `part2.py` through Autolab. We will test your code on several problems, including on the grid world example given in AIMA(4ed) on pg. 646, as above.

Part 3: Multi-Armed Bandit

First mentioned in the foundation book, Reinforcement Learning: An Introduction, by Sutton and Barto, the multi-armed bandit is a simple and classic RL problem. An agent is free to choose any action in a state independent environment, where the reward for each action is given according to an underlying probability distribution. This situation can be best compared to a casino with multiple slot machines such that each one has its own probability distribution of success. The fact that we do not have access to this probability distribution is what makes this problem non-trivial. This problem brings up the classic exploration vs. exploitation tradeoff. Your task is to:

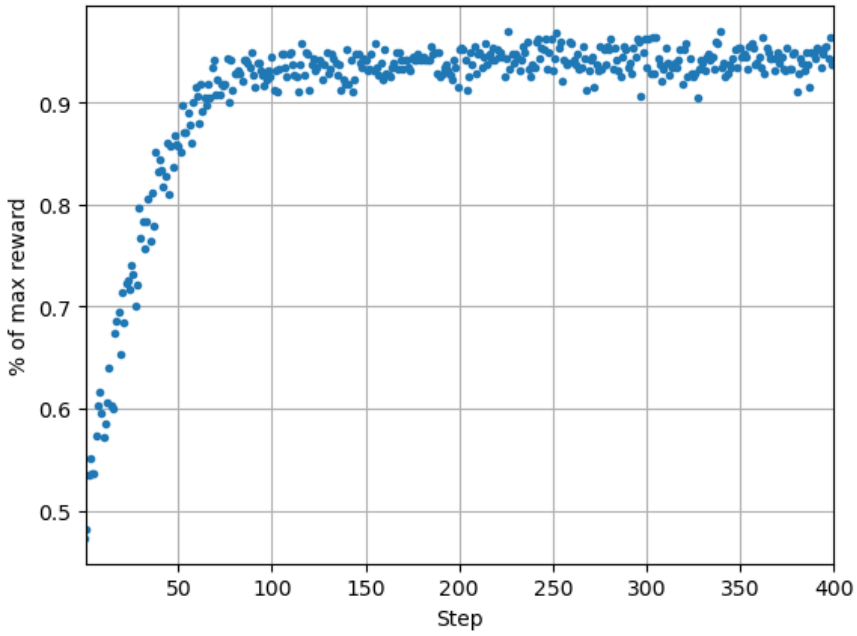

Figure 2: The received reward averaged over 500 experiments for each execution step of a RL policy on the MAB environment.

1. Implement an agent in `part3.py` to solve the multi-arm bandit (MAB) problem. The provided class methods `update_state` and `get_action` are used by the testing scripts, as shown in `test_part3.py`, and hence their method signatures must be kept compatible. You are allowed to implement additional class methods for better code organization.

Each arm in this problem provides a reward of +1 with an independent probability, and 0 otherwise. The goal of the RL agent is to maximize the received reward as fast as possible while it picks a slot machine arm one at a time. Figure 2 shows a plot of the received reward as an efficient RL agent executes its policy across 400 episodes.

Grading

We would like to reiterate that **only** the functions implemented in the template files will affect your marks. In order to make this task more challenging, a submission threshold of **14** has been set on Autolab. Given the stochastic nature of the problem setup, the RL policy in part 3 will give a different result on each execution — we recommend that you save the last 4 submission to maximize marks on your final 10th submission. For this project, Autolab has a timeout of **600 seconds** per submission. Points for each portion of the project will be assigned as follows:

- Value Iteration – **20 points** (4 test × 5 points each)

  The first test (Part 1A) will evaluate your state transition model, to ensure that you understand how to write down such models from a high-level description (i.e., that given in the caption to Figure 1). The next three tests (Part 1B) will evaluate your value-based MDP solver, first for the simple 1-D cleaning robot world, and then for the grid world defined in AIMA (with some tweaks).

  Time limit: 1 second per test.

- Policy Iteration – **10 points** (2 tests $\times$ 5 points each)

  The two tests will evaluate your policy-based MDP solver on different types of grid worlds.

  Time limit: 1 second per test.

- Multi-Armed Bandit – **20 points** (2 tests $\times$ 10 points each)

  Your RL agent will be run on a random MAB environment for 400 episodes. This experiment will be run 500 times and the average received reward over the episodes will be evaluated. The goal of this task is to receive a final reward that is as close as possible to the maximum possible reward, and reach this value with the fewest episodes. For example, the RL agent in figure 2 is able to achieve more than 90% of maximum reward within 100 episodes. Two tests will evaluate your episode execution time as well as your reward learning curve. The marking scheme for the first test on Autolab is provided as a comment in `test_part3.py` for reference and further clarity.

  Time limit: to ensure that the experiments finish running within the Autolab time limit, each episode must take less than 0.001s to run. See `test_part3.py` for details on how this is enforced.

Total: **50 points**

Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation, and code commenting. Please note that we will test your code *and it must run successfully*. Code that is not properly commented or that looks like 'spaghetti' may result in an overall deduction of up to 10%.