Advanced Machine Learning Assignment — 2023

Submission

Please submit your solution electronically via vUWS. (1) Submit a report as PDF via Turnitin. (2) Create a `zip` file with your code (use zip, do not use `rar`), and any other file you want to submit, and upload it to vUWS (to where you got this assignment text), and please **include the signed and completed cover sheet that you can find at the end of the document**.

Submission is due on 26 Oct 2023, 11:59pm.

Robot locomotion

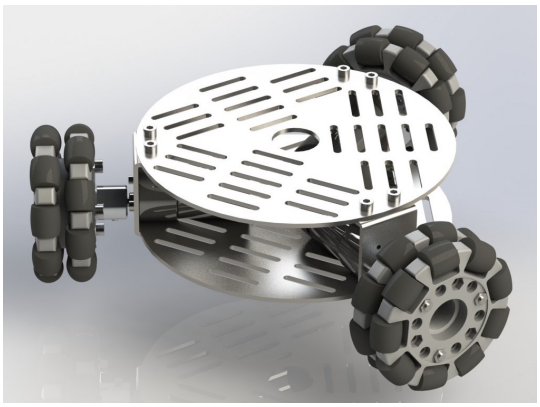

Figure 1: Rendering of an omniwheel robot (left) (credit: Thang Trinh, `https://grabcad.com/library/robot-omni-3-wheel-1`), and a screenshot of our simulation (right)

In this assignment we work with a simulation of an omniwheel robot. The robot has 3 wheels, and can be moved by setting the velocity of each wheel (between -1 and 1). Because of its special wheels, the robot can move in any direction and also spin on the spot or while it is moving.

Preparation Download the `omniwheel3a.py` and `omniwheel3b.py` python files. They both implement the same robot simulation (using `PyGame`, see `https://www.pygame.org/wiki/GettingStarted` if you need help installing it). The difference between the two files is that in the first one, the robot has to reach a static target, and in the second one the robot needs to get to a moving target.

At the beginning of an episode, the robot and the target appear at a random location and orientation on the screen. At each step, the robot can set the velocities of its 3 wheels ω_i to a value between -1.0 and 1.0 (inclusive). This velocity applies instantaneously.

The goal is to get the robot to the target (an orange ball), for which a reward of +10 is given. For every step the robot has not reached the ball yet, it will receive a small negative reward of -0.01. An episode finishes when the robot successfully gets close to the ball, or after 1000 unsuccessful steps. An episode also finishes when the robot leaves the area, leading to an additional negative reward. In the files on vUWS, the robot takes random actions. Your task is to replace these lines and add code for reinforcement learning in appropriate places.

Task 1: Reaching a static target, tabular RL 15 points

For this task, use the python program `omniwheel3a.py`. The robot pose (x, y, θ) describes the position and the orientation of the robot. The purple wheel is considered the rear of the robot: $\theta = 0$ means the robot points upwards (purple wheel towards the bottom of the screen).

To create a tabular RL program that aims to move the robot close to the ball, you can make use of the robot pose (x, y, θ), and `target_pos` (the x- and y position of the ball). To represent all possible locations and orientations in a table would result in a very large table, making learning the best actions difficult. Also, an action for the robot consists of 3 continuous values between -1 and 1 (one value for each motor). A first step, therefore, is to simplify the task for the robot, and for you. A second step is to train this simplified task using a tabular RL approach. The third step is to evaluate your final policy. You will need to submit both text (as part of your report) and your implementation (python code, as a separate `.py` or `.ipynb` – not just as part of your report).

1. Reducing the number of states: The robot has access to its own pose (x, y, θ), and the target position (x- and y-coordinate). Create a python function that converts the robot pose and the target position into a small number of states. Here is one possible example that you could implement:

State	Description
0	ball in front of the robot
1	ball left in front of the robot
2	ball right in front of the robot
3	ball left behind the robot
4	ball right behind the robot

You could reduce the number of different states even more, use a larger number, or use a different representation.

For tabular RL, we also would like to reduce to a small number of actions. You can create a small number of actions by considering only 3 possible actions for each motor: $\omega_i \in \{-0.5, 0.0, 0.5\}$. The combination of these 3 values for the 3 wheels will give you 27 possible actions in total (you can take a different number of values, or different velocities, if you prefer).

For this step, implement and submit a python function that computes a state from a given robot pose and ball position. Describe the different states, and how you compute them. Submit both the description and your python code (of the function that computes the state). If you try multiple different approaches, submit the one that worked best for you, but you can describe what effect the change on the learning had. With your chosen representation and the number of possible actions that you selected: what will be the size of your tabular policy?

2. Now that you have a simplified representation, implement a tabular RL approach. You can choose a to learn a deterministic or a stochastic policy as you prefer. When you run the training, collect the total reward for each episode, so that you can create a plot of the rewards over time. If you decide to use ϵ-greedy action selection, set $\epsilon = 1$, initially, and reduce it during your training to a minimum of 0.01. Keep your training going until you are either happy with the result or the performance does not improve[1]. After or during the training, plot the total sum of rewards per episode. This plot — the *Training Reward* plot — indicates the extent to which your agent is learning to improve its cumulative reward. It is your decision when to stop training. It is not required to submit a perfectly performing agent, but show how it learns. Also submit the code that you implemented for this step (the training approach and loop, including the code to create the plots).

3. After you decide the training to be completed, run 50 test episodes using your trained policy, but with $\epsilon = 0.0$ for all 50 episodes. Calculate the average over sum-of-rewards-per-episode (call this the *Test-Average*), and the standard deviation (the *Test-Standard-Deviation*). These values indicate how your trained agent performs; include them in your report and describe the approach and the behaviour it leads to. Also submit the code that you implemented for this step (the test loop).

Task 2: Train to approach a moving target

10 points

For this task, use the python program `omniwheel3b.py`. The environment is the same with the only difference that the target is now moving.

[1] This means: do not stop just because ϵ reached 0.01 – you may want to stop earlier, or you may want to keep going, just do not reduce ϵ any further.

As a first step, create a simple representation that includes information about the target velocity suitable for a tabular approach. In a second step, train your approach for this environment. As a third step, test your newly trained approach, and compare it against the trained approach for the static environment in `omniwheel3a.py` but applied to the dynamic version.

1. Reducing the number of states: The robot has access to its own pose (x, y, θ), and the target position (x- and y-coordinate), and the target velocity. Create a python function that converts the robot pose, the target position, and the target velocity into a small number of states.

2. Train your approach from task 1 for this environment, with your new representation. Create the *Training Reward* plot similar to the first task, and compare the two different scenarios in your report (training times, average rewards). Also submit the code that you implemented for this step (the training approach and loop, including the code to create the plots).

3. Similar to above, run 50 test episodes using your trained policy, but with $\epsilon = 0.0$ for all 50 episodes. Again calculate the *Test-Average*), and the *Test-Standard-Deviation* for your new agent. Also run 50 test episodes of your first policy on this environment, compute the test-average and test-standard-deviation. Compare the numbers and the behaviour of your robots. Also submit the code that you implemented for this step (the test loop).

Task 3: Moving Target, Policy Gradient Methods 15 points

In this task, you will apply a policy gradient method (for example, REINFORCE) to train the robot. The environment is the same as in Task 2, with the target dynamically moving. Unlike in the tabular approach, you will be using a neural network to represent your policy.

1. Network Architecture: Design a neural network architecture suitable for the policy representation. Explain your choice of architecture, including the number of layers, nodes in each layer, and activation functions. Our suggestion is to keep the neural network small and simple to avoid long training times. Your first goal is to show how the approach would be applied, not necessarily to achieve the best possible performance.

2. Training Setup: Describe the policy gradient algorithm you have chosen, and explain your choice. Configure your algorithm's hyperparameters such as learning rate, discount factor, and any other relevant parameters.

3. Training: Train your agent using the policy gradient method. Generate the *Training Reward* plot to display the learning progression. The plot should show the

total reward earned per episode over time. Discuss any patterns or anomalies you observe. Also, submit the code that you implemented for this step (the training approach and loop, including the code to create the plots).

4. Evaluation: After training, run 50 test episodes using your trained policy. Calculate the *Test-Average* and the *Test-Standard-Deviation*. Include these metrics in your report along with a description of how well your policy performs. Also submit the code that you implemented for this step (the test loop).

5. Comparison: Compare the performance and training time of your policy gradient approach with the tabular methods you implemented in Tasks 1 and 2. Discuss any trade-offs, benefits, or insights you gained from using a policy gradient method.

Task 4: Predicting Steps to Target Using Neural Networks 10 points

The goal of this task is to design and implement a neural network that predicts the number of steps a trained RL agent from either Task 1, 2, or 3 would take to reach the target ball. This is a regression problem, and you will be using supervised learning to accomplish this.

1. Data Collection: Use the trained agents from Tasks 1, 2, or 3 to collect data. For each episode, log the initial robot pose, initial target position, and the number of steps taken to reach the target. Store this data in a suitable format for training your neural network.

2. Network Architecture: Design a simple neural network suitable for the regression task. Describe your choice of architecture including the number of layers, nodes in each layer, and activation functions.

3. Training: Train the neural network on the collected data. You can split the data into training and validation sets to tune the model. Discuss your choice of loss function, learning rate, and any other relevant hyperparameters. Also, submit the code for this training step.

4. Evaluation: Use a separate test set to evaluate the performance of your trained model. Calculate metrics such as Mean Absolute Error (MAE) or Root Mean Square Error (RMSE) to quantify the model's performance. Include these metrics in your report, along with any observations about the model's accuracy.

Tips

1. For the RL-tasks, it often takes some time until the learning picks up, but they should not take hours. If the agent doesn't learn, explore different learning rates. For Adam, start with values between `5e-3` (faster) and `1e-4` (slower).
2. Even if the learning does not work, remember that we would like to see that you understood the ideas behind the code. Describe the ideas that you tried, and still submit your code but say what the problem was.

You *can* ask or answer questions about how to *use* the files provided with this assignment online, as long as they are general python / programming questions, for example if the code provided does not work for you as expected. You must not ask or answer questions to the machine learning questions in this assignment anywhere, including chatGPT. If in doubt, ask your friendly lecturers or tutor first.

Assignment Cover Sheet

School of Computer, Data, and Mathematical Sciences

Student Name	
Student Number	
Unit Name and Number	INFO7001: Advanced Machine Learning
Title of Assignment	Assignment 1
Due Date	26 Oct 2023
Date Submitted	

DECLARATION

I hold a copy of this assignment that I can produce if the original is lost or damaged.

I hereby certify that no part of this assignment/product has been copied from any other student's work or from any other source except where due acknowledgement is made in the assignment. No part of this assignment/product has been written/produced for me by another person except where such collaboration has been authorised by the subject lecturer/tutor concerned.

Signature: . . . . . .

(Note: An examiner or lecturer/tutor has the right not to mark this assignment if the above declaration has not been signed)

	Task 1	Task 2	Task 3	Task 4	Total
Mark					
Possible	15	10	15	10	50

The maximum points possible for this assignment is 50.