CO424 Reinforcement Learning Part 2: Coursework Part 2

Dr. Edward Johns

Wednesday 13th November 2019

This is Part 2 of the coursework for the second half of CO424 Reinforcement Learning. My advice is to complete Part 1 of the coursework, before moving on to Part 2. **The deadline for both parts of the coursework is 7pm on Thursday 21st November**.

Files

Take a look at the Python files *random_environment.py*, *agent.py*, and *train_and_test.py*. The relevant parts of the code are well commented, so you may be able to understand these files by reading through them. Below is a short description of each file.

random_environment.py
This is similar to the *environment.py* file in Coursework Part 1. The difference now is that the environment it creates is more complex, and it creates a random environment every time the *Environment* class is instantiated.

agent.py
This is the only file which you should need to edit. There are currently five functions, which are used by *train_and_test.py* to allow the agent to interact with the environment. You should ensure that the names of these functions, and their argument lists, do not change. However, you are free to edit the code within each of these functions.

train_and_test.py
This is the script which will be used to train, and then test, your agent. It will create a new random environment, train your agent for 10 minutes in this environment, and then test your agent's greedy policy in this environment with an episode of 100 steps.

If you run *python3 train_and_test* from the command line, you should see the environment being displayed. The red circle is the agent's current state, and the green circle is the goal. The black region is free space, which the agent can move through, and the grey region is an obstacle, which the agent cannot move through. If the agent tries to move into the obstacle, the agent will remain in its current state. The agent must navigate the "maze" and reach the goal as quickly as possible.

Implementation

Your task is to create your own deep Q-learning implementation in the file *agent.py*. Your implementation may be similar to the code you have written for Coursework Part 1, and there are various hyperparameters which you can experiment with. See the end of the slides for Lecture 2 for a list of these hyperparameters. You are also free to implement any of the advanced deep Q-learning methods which were introduced in this course. Again, see the end of the slides for Lecture 2 for a list of these advanced methods. Note that when your code is evaluated, it will be trained for 10 minutes; so some hyperparameters, or advanced methods, may slow down the training too much to achieve good performance within this time.

I suggest proceeding in the following order:

1. Take the code you have developed in Coursework Part 1, and rewrite it so that it fits into the structure of the new Agent class.
2. Optimise the hyperparameters, and find a set of hyperparameters which work well for a range of different environments.
3. Introduce some of the Deep Q-learning extensions listed above, and see if they improve the agent's performance. The above hyperparameters may need to be optimised again for each extension you introduce.

Remember that for this course, 50% of the grades are awarded for your courseworks. This is larger than most courses, and so you are expected to spend more time on coursework in this course, than in other typical courses. Part 2 of the coursework is advanced and is intended to challenge even the top students. If you are having difficulty, just focus on implementing the basic deep Q-learning implementation from Coursework Part 1, and you will still be awarded decent grades.

Rules

Below are some rules which your solution must abide by:

- The magnitude of your action vector must not exceed 0.02. Otherwise, the agent will stay still. You can see this imposed in line 104 of *random_environment.py.*
- Your *agent.py* may import any Python 3 modules from the Python 3 standard library. However, it may only import the following additional Python modules: numpy and torch.
- Your implementation must be a genuine deep Q-learning solution. You may not use tabular reinforcement learning. You may not create a hand-crafted controller (e.g. keep moving right, if the agent hits a wall, randomly move up or down until it can move right again, repeat ...). To check: if you have a neural network trained to predict one or multiple Q-values, and your *Agent.get_greedy_action()* returns the action with the highest Q-value based on this neural network, then your solution is acceptable. Otherwise, your solution is not acceptable.
- You may not import anything from the *environment* module into the *agent* module. For example, you might be tempted to import the goal state, or the locations of the obstacles. This is not allowed; the agent must learn only from the *distance_to_goal* value which is returned from the environment. Importing the environment will not work anyway, since the *environment.py* file we test with is different to the one you have been provided.
- You may not copy and paste entire existing implementations you have found elsewhere. Any implementations must be your own, and we will be checking your code for plagiarism. Feel free to use other implementations for inspiration, but do not copy and paste.

Submission

You should submit two files to CATE. First, is your *agent.py* file. You should name this file *agent_123456.py*, where *123456* are the last 6 digits of your CID number. Second, is a one-page PDF. You should name this file *part_2_123456.pdf*, where *123456* are the last 6 digits of your CID number. This is in addition to the PDF file you will submit for Coursework Part 1 (*part_1_123456.pdf*). So in total, this is three files: two .pdf files, and one .py file.

Your *agent_123456.py* will contain your deep Q-learning implementation. When we receive this file, we will train and test your agent using the script *train_and_test.py.* Therefore, you need to make sure that your *agent_123456.py* code is compatible with *train_and_test.py.* For example, if you run *train_and_test.py* with the original *agent.py*, you will see that the agent moves randomly around the environment during training, and will then move to the right during testing. After this, the result of the test will be printed out. You need to make sure that your *agent_123456.py* file will also allow *train_and_test.py* to run the training and testing in this way. So, be sure that you have run *train_and_test.py* fully, both for training and testing, to ensure that your *agent_123456.py* file will run as expected. To do this, in *train_and_test.py* line 5, you can replace *from agent import Agent* with *from agent_123456 import Agent*.

If there are any bugs in your *agent_123456.py*, such that running *train_and_test.py* cannot train or test your agent, then you will have marks deducted; we will not spend time fixing your code for you. **Important:** It is better to have a simple solution that runs fully, rather than a complex solution which contains a minor bug, causing the training or testing to not run fully. Please also ensure that you have tested your code using Python 3.7, and the Python packages listed on GitLab here.

Your *part_2_123456.pdf* file should be a one-page A4 document with size 12 font, with your name and CID number written at the top, and containing a description of your final method implemented in *agent_123456.py*. You are free to write what you want in this document, but it should clearly and concisely explain what you have implemented. You may wish to focus on certain aspects of your implementation if you think they are particularly interesting. You may wish to explain why you believe certain hyperparameters work best, or why certain methods you have tried worked better than others.

Important: When describing your implementation, use line numbers (from your *agent_123456.py* file) to help you describe what you have implemented. During marking of the document, methods you describe will be verified in your *agent_123456.py* file. Therefore, if you do not include line numbers to make the relevant code sections easy to find, you may lose marks for a lack of clarity. You do not need to describe every single line of your code, but you should be able to write down the line number ranges for each section which you are describing in the document. Even if your implementation is effectively the same implementation as Part 1 of the coursework, you should still describe all components, and in which lines of your code they appear. You may wish to add comments to your code, but you should not rely on these in your document. For example, you cannot write "See the comments in lines 30-40" in your document in order to save space.

Grading

Coursework Part 2 contributes towards 40% of the grade for the coursework in the second half of this course. This equals 20% of the total coursework across both halves, or 10% of the entire course. 50% of your marks for Coursework Part 2 will be awarded for your report, and 50% will be awarded for the score achieved when your code is tested.

For your report, you will be assigned grades for: clarity in the description of your method, and sophistication of your implementation. But sophistication does not mean complexity. A neat, concise implementation, following the material in the lectures, will be awarded higher grades than a very long, chaotic implementation. **Important**: You will be awarded more grades for a simple solution which achieves a decent score in the evaluation, than a complicated solution which scores poorly in the evaluation.

For evaluation of your code, *train_and_test.py* will be used to train your agent for 10 minutes, and then test your agent using its greedy policy. This evaluation will be done on an Intel Core i7-7820X CPU. You will receive a grade based on how close your agent is to the goal after an episode of 100 steps. If your agent reaches a distance of less than 0.03 to the goal at any point, then you will receive bonus grades, based on how many testing steps it took the agent to reach the goal. Your code will be evaluated on three different, random environments, with grades averaged across the three. Therefore, you should ensure that your method is able to train the agent well across a range of different environments.

Good Luck, and Have Fun!