COMP3702 Artificial Intelligence (Semester 2, 2024)
Assignment 3: Reinforcement Learning

Key information:

- **Due: 1pm, Friday 25 October**
- This assignment assesses your skills in training and understanding algorithms for solving Reinforcement Learning Problems.
- Assignment 3 contributes 20% to your final grade.
- This assignment consists of two parts: (1) programming and (2) a report.
- This is an individual assignment.
- Your program (code) must be zipped and submitted to Blackboard. It will not be explicitly graded, and instead will form the basis for analysis in the report.
- Your report should be in .pdf format and named according to the format `a3-COMP3702-[SID].pdf`, where SID is your student ID. **Each question should start on a separate page and be clearly indicated. Failure to do so will result in a 5% penalty**. The report is to be submitted via Gradescope. Reports will be graded by the teaching team.
- **Note: this assignment will take several hours to run. Ensure you start early!**

Reinforcement Learning

In this assignment, you will implement Deep Reinforcement Learning algorithms and analyse their parameters and performance. This assignment will test your skills in training and understanding reinforcement learning algorithms for practical problems and understanding of key algorithm features and parameters.

Gymnasium API

This assignment will make use of the OpenAI Gym/Gymnasium[1], which is a standard API for reinforcement learning with a diverse collection of reference environments. This assignment will start by investigating the Cart Pole and Lunar Lander environments shown in Figure 1, which are within the `classic-control` and `box2d` set of environments respectively. If using pip, you can install Gymnasium and the dependencies for these set of environments via the following:

```
pip install gymnasium
pip install gymnasium[classic-control]
```

Additional dependencies are required in some environments as per the documentation: e.g. `pip install swig` for Box2D environments and `pip install pygame` for the visualisation libraries.

Figure 1: Cart Pole and Lunar Lander environments

[1] Gymnasium is a maintained fork of OpenAI's Gym library.

To get an understanding of these environments, you can visualise them using the human render mode using code as below:

```
import gymnasium as gym

# Initialise the environment
# You can replace the environment e.g. "LunarLander-v2" or -v3
# Ensure the environment (e.g. classic-control, box2d) is installed using pip/conda
env = gym.make("CartPole-v1", render_mode='human')

# Reset the environment to generate the first observation
observation, info = env.reset(seed=42)
for _ in range(500):
    # this is where you would insert your policy, here we randomly sample an action
    action = env.action_space.sample()

    # step (transition) through the environment with the action, receiving the
    # next observation, reward and if the episode has terminated or truncated
    observation, reward, terminated, truncated, info = env.step(action)

    # If the episode has ended then we can reset to start a new episode
    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

PyTorch

To train neural networks, we will be using the PyTorch machine learning framework. To download PyTorch, follow the instructions here: https://pytorch.org/get-started/locally/, following the instructions for your computer system.

If your computer has a dedicated GPU, select PyTorch with GPU support, and download the appropriate drivers (ROCm for AMD GPUs and CUDA for Nvidia GPUs). Make sure you only install one - either the GPU or the CPU version! Note that the classic-control environments can train faster using CPU than GPU.

Task

Your task is to run the Deep Q-Network (DQN) algorithm and variants such as Duelling DQN and Double DQN, and to write a report investigating the algorithms' performance and hyper-parameters as detailed in the report section.

For background, we recommend that you read through the following tutorials. You can make use of the code in your solutions with attribution:

- Official DQN PyTorch Tutorial: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
- Tutorial sample code: https://github.com/comp3702/tutorial11

When training reinforcement learning algorithms, typically we assess the solution quality using the 100-step moving average episode reward (i.e., **R100**) received by the learning agent. At time step t, the 100-step moving reward is the average episode reward earned by your learning agent in the episodes $[t-100, t]$. If the Q-values imply a poor quality policy, this value will be low. If the Q-values correspond to a high-value policy, the 100-step moving average reward will be higher. We use a moving average because rewards may only be received occasionally and the episode reward is affected by sources of randomness including the exploration strategy. You will need to write a function that plots the R100 vs episodes for analysis in the report.

The report

The report tests your understanding of Reinforcement Learning algorithms and the methods used in the code. You should use the code at https://github.com/comp3702/tutorial11 as a foundation to complete the assignment.

Question 1. Q-learning vs Value Iteration
Q-learning is closely related to the Value Iteration algorithm for Markov decision processes.

a) Describe two **key** similarities between Q-learning and Value Iteration. Answer in no more than 5 lines of text. (5 marks)

b) Give one **key** difference between Q-learning and Value Iteration. Answer in no more than 5 lines of text. (5 marks)

Question 2. Comparing CartPole-v0 and CartPole-v1
In this question, you will train two DQNs for two versions of CartPole – v0 and v1 – till the R100 value reaches the `reward_threshold` (or "stopping_reward") and compare the two learnt policies. According to the source code, the difference between CartPole-v0 and CartPole-v1 are the `max_episode_steps` and `reward_threshold` as specified in Table 1.

Table 1: Differences between CartPole-v0 and -v1

Environment	`max_episode_steps`	`reward_threshold`
CartPole-v0	200	195.0
CartPole-v1	500	475.0

An episode ends if any one of the following occurs:

- Termination: Pole Angle is greater than $\pm 12°$
- Termination: Cart Position is greater than ± 2.4 (centre of the cart reaches the edge of the display)
- Truncation: Episode length is greater than 500 (for CartPole-v1) or 200 (for CartPole-v0)

a) Implement a function to plot the R100 value vs Episode number. You will need to import a plotting library, e.g. `import matplotlib.pyplot as plt`, and can implement a function similar to that used in the PyTorch tutorial. Copy or screenshot your code implementation for your answer, citing any resources you used to develop this. (As part of this, you may also want to implement saving and loading of results and/or plots). (5 marks)

b) Plot the R100 value vs Episode number for CartPole-v0 and CartPole-v1 DQN models. Ensure your axes are correctly labelled and indicate what each plot represents (e.g., using a legend or caption). (5 marks)

c) Describe and compare the learnt policies for CartPole-v0 and CartPole-v1. You may make use of the saved video examples on Blackboard titled "CartPole-v0.mp4" and "CartPole-v1.mp4". Based on your observation of these learnt policies, the definition of the environment and your plots, explain why you think the values of `max_episode_steps` and `reward_threshold` were increased from v0 to v1. (5 marks)

Note: you may need to train the model several times to observe the desired behaviour differences. You can use the human render mode to visualise the policies extracted from the trained neural networks as in the Tutorial solutions or simply describe the saved videos on Blackboard.

Question 3. Loss function and Target network

When training neural networks, a loss function is used to compare the current neural network's predictions to ground truth values, and then gradient descent with backpropagation is applied to adjust the weights of the neural network so the predictions move closer to the ground truth values. In supervised learning we make use of a dataset of paired input and output labels $\{x, y\}$ for training. In reinforcement learning, the agent receives experiences of states, actions and rewards and through exploration it must estimate the state-action values (in Q-learning).

a) With reference to TD-learning, describe the loss function used to train the neural network in DQN. Use equations and highlight the components corresponding to the "target value" and the neural network's current state-action value predictions. (5 marks)

b) Examining the code in `dqn_gym.py`, specify which variables correspond to the "target values" and the current state-action value predictions/estimates. (5 marks)

c) An important implementation detail in DQNs is having a separate target network. Describe why a separate target network is needed. (5 marks)

d) Compare the performance of synchronising the target network periodically (e.g. `alpha_sync` = false, `target_net_sync` = 1000), versus using soft updates (e.g. `alpha_sync` = true, `tau` = 0.005). You may select which environment to apply this to (e.g. CartPole-v0). The parameters can be updated in a config file e.g. `dqn.yaml` (10 marks)

Question 4. Learning-rate

For this question, consider the CartPole-v0 environment (or an environment of your choice) and use the R100 value (100-step moving average episode reward) as a measure of the quality of the learnt policy.

a) Plot the quality of the policy learned by DQN, as given by R100, against episode number for three different fixed values of the `learning_rate` (which is called α in the lecture notes and in many texts and online tutorials). For this question, do not adjust α over time, rather keep it the same value throughout the learning process. Your plot should display the solution quality up to an episode count where either the performance stabilises (typically > 1000 episodes) or a clear difference in learning rates can be observed. (5 marks)

b) With reference to your plot(s), comment on the effect of varying the `learning_rate`. (5 marks)

c) Use a plot (either self-drawn or sourced and cited) to describe what happens when the `learning_rate` is too high. (5 marks)

Question 5. Epsilon

a) Describe the purpose of the epsilon hyperparameter. (5 marks)

b) Briefly explain how the values of `epsilon_decay`, `epsilon_final` and `epsilon_start` affect training performance. (Note: You do not need to show plots in your answer, but will need to do experimentation to understand the effect of the parameters). (5 marks)

Question 6. DQN vs Double DQN or Duelling DQN

Two variants of DQN are Double DQN and Duelling DQN. Details of the methods can be read in the papers:

- Double DQN: https://arxiv.org/abs/1509.06461
- Duelling DQN: https://arxiv.org/abs/1511.06581

An implementation of Duelling DQN is included in the tutorial11 code.

Choose one of these algorithms and discuss its improvements relative to the original/vanilla DQN. (10 marks)

Question 7. Applying DQN beyond CartPole (20 marks)

Based on your study of hyperparameters and understanding of DQN, train a DQN agent for an additional environment from Gymnasium and report on any implementation/hyperparameter changes you had to make, and your agent's performance. For a simple environment consider LunarLander-v3 or MountainCar-v0. For environments that are image-based like Pong, you will need to use convolutional neural networks, and will require a GPU for efficient training. See sample code here: https://github.com/comp3702/dqn-pong.

This question is effectively repeating the experiments from the previous few questions, but where you get to choose which experiments to perform (e.g. which hyper-parameters to vary), and for your chosen environment instead of CartPole. e.g. choose some hyper-parameters (learning rate, epsilon, tau/target sync interval, number of hidden layers, etc), try a few values for each, and select the best value for each of your chosen hyper-parameters based on your results (which allows you to justify your choice of those values).

Criteria:

- Experiment with ≥ 3 hyperparameters and provide evidence and justification for your selection
- Provide evidence of experimentation including plots comparing performance of various hyperparameter settings
- Describe what you observed / the effect of the parameter change and why you chose it/ why you modified it
- Report final settings of hyperparameter values
- Demonstrate that you can solve a level. e.g. define what is a good policy/solved for your selected environment and show that your agent achieves this. e.g. R100 value or description/screenshot of behaviour

Ensure you use headings and paragraphs and tables where appropriate, to facilitate reading of your assignment.

Bee Cart Pole credit to Anonymous COMP3702 student and Generative AI.

END OF ASSIGNMENT 3

Academic Misconduct

The University defines Academic Misconduct as involving "a range of unethical behaviours that are designed to give a student an unfair and unearned advantage over their peers." UQ takes Academic Misconduct very seriously and any suspected cases will be investigated through the University's standard policy (https://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct). If you are found guilty, you may be expelled from the University with no award.

It is the responsibility of the student to ensure that you understand what constitutes Academic Misconduct and to ensure that you do not break the rules. If you are unclear about what is required, please ask.

In the coding part of COMP3702 assignments, you are allowed to draw on publicly-accessible resources and provided tutorial solutions, but you must make reference or attribution to its source, in comments next to the referenced code, and include a list of references you have drawn on in your `solution.py` docstring.

If you have utilised **Generative AI** tools such as ChatGPT, you must clearly cite any use of generative AI in each instance. To reference your use of AI see:

- https://guides.library.uq.edu.au/referencing/chatgpt-and-generative-ai-tools

Failure to reference use of generative AI tools constitutes student misconduct under the Student Code of Conduct.

It is the responsibility of the student to take reasonable precautions to guard against unauthorised access by others to his/her work, however stored in whatever format, both before and after assessment. You must not show your code to, or share your code with, any other student under any circumstances. You must not post your code to public discussion forums (including Ed Discussion) or save your code in publicly accessible repositories (check your security settings). You must not look at or copy code from any other student.

All submitted files (code and report) will be subject to electronic plagiarism detection and misconduct proceedings will be instituted against students where plagiarism or collusion is suspected. The electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. If you collude to develop your code or answer your report questions, you will be caught.

For more information, please consult the following University web pages:

- Information regarding Academic Integrity and Misconduct:
  - https://my.uq.edu.au/information-and-services/manage-my-program/student-integrity-and-conduct/academic-integrity-and-student-conduct
  - http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct
- Information on Student Services:
  - https://www.uq.edu.au/student-services/

Late submission

Students should not leave assignment preparation until the last minute and must plan their workloads to meet advertised or notified deadlines. It is your responsibility to manage your time effectively.

It may take the autograder up to an hour to grade your submission. It is your responsibility to ensure you are uploading your code early enough and often enough that you are able to resolve any issues that may be revealed by the autograder *before the deadline*. Submitting non-functional code just before the deadline, and not allowing enough time to update your code in response to autograder feedback is not considered a valid reason to submit late without penalty.

Assessment submissions received after the due time (or any approved extended deadline) will be subject to a late penalty of 10% per 24 hours of the maximum possible mark for the assessment item.

In the event of exceptional circumstances, you may submit a request for an extension. You can find guidelines on acceptable reasons for an extension here https://my.uq.edu.au/information-and-services/manage-my-program/exams-and-assessment/applying-extension, and submit the UQ Application for Extension of Assessment form.