IM707 Deep Reinforcement Learning Coursework

Yuichi Kuriyama
Stefan Diener

Basic Task

1. Define an environment and the problem to be solved

The aim of this basic task is to implement the Q-learning algorithm and apply it to a reinforcement learning problem. In order to address this, the Open AI platform [1] has been chosen as a suitable toolkit. The Open AI platform is an open source interface toolkit which allows the development of reinforcement algorithms by offering a wide range of diverse problems. Given the suitability and scope of this project, the Cart-Pole problem originally developed by Barto, Sutton, and Anderson [2] has been chosen as a task for this section.

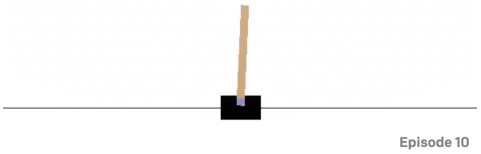

Figure 1: CartPole-v1 [1]

Figure 1 shows a snapshot of one of the episodes of this task as an example. The goal of this task is to balance the pole attached by an un-actuated joint cart and prevent the pole from falling down whilist sustaining the pendulum upstraight. Firstly, an initial position can be defined by the uniformed random value ranging from -0.05 to 0.05. As a possible action, forces have been applied +1 (right) and 0 (left) to indicate the direction . After 100 time steps and an average reward of 195, the task is considered as solved. Episodes will terminate if the Cart Position is greater than +/- 2.4, the Pole Angle is greater than +/-12° or the episode length is more than 200. The definition of solving this task is to obtain an average reward of 195 over 100 trials.

2. Define a state transition function and the reward function

2-1 State transision function

Based upon the possible action, the state can be determined by the car position, car velocity, pole angle and pole angular velocity. Observsation space was defined as follows:

Num	*Observation*	*Min*	*Max*
0	*Car Position*	*-4.8*	*+4.8*
1	*Catt Velocity*	*-inf*	*+inf*
2	*Pole Angle*	*-24°*	*24°*
3	*Pole Angular Velocity*	*-inf*	*+inf*

Table 1: Observation space

As the observation value is an infinite and continuous value, which cannot be expresse in a resonably sized Q-table. In order to mitigate this risk, we cut the observed space into 30 bins to convert from continuous to discrete values. The state transition fuction is defined as follow:

$$S_{next} = t(s, a)$$

Equation 1

Next state S_{next} is determined by transition function of the current state and the action a picked up by the current state.

2-2 Reward Function

As mentioned earlier, the goal of this task is to keep the cartpole straight as much as possible. In this case study using Open-AI environment, we use the in-build reward. Therefore, a reward of +1 will be given for each time step where the experiment is running and no termination rules have been triggered. Likewise, the reward function is defined as follows:

$$r' = r(s, a)$$

Equation 2

This illustartes a basic structure of reward function, showing that r is reward given by taking an action a in state s.

3. Set up the Q-learning parameters (gamma, alpha) and policy

As an intial experiment, the Q-learning paremeters were set as follws:

Learning rate	0.25
Gamma	0.99
Episode	10000
Time step	100
Starte Epsilon	0.2

Table 1

Learning rate can be used to determine the magnitute of the update applied to the q values. The large learning rate might not converge to the global minima and in contrast, the small learning rate might take a considerable amount of time to reach its minima. In this case study, the learning rate is set as 0.25 although more investigation by utlising the hyper-parameter tuning will be conducted in the later section. Gamma, the so-called discount factor ranging from 0 to 1 is used to determine the importance of future rewards. The closer to 1, Q-learning model will aim to prioritize the future rewards as opposed to the immediate rewards. This would reduce the noise reward gained by random chance and place more importance upon intended actions. In order to observe the meaningful performance, the algorithm was run for 10000 episodes. Furthermore, epsilon-greedy policy was applied to make adjustments to the balance between the exploration and exploitation problem [3].

Exploration enables agents to update the knowleadge by investigating new action values for the sake of the future reward and exploitation allows them to obtain the most reward by applying a greedy approach.

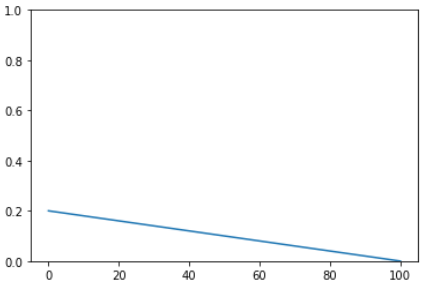

Figure 2

In this experiment, if the random number is less than epsilon, the agents will choose a random action. In order to priortise exploitation in the later learning process, epsilon starting from 0.2 will be decreased linearly in line with the number of timestamps, which comprise 100 episodes each, as shown by figure 2.

4. Run the Q-learning algorithm and represent its performance

4-1 Q-learning algorithm

In this task, in order to solve the problem, Q-learning will be applied to the environment. Q-learning is an off-policy learning to separate the deferral policy from the learning policy and update the action selection using the Bellman optimal equations [4] by representing Equation 1 (noted Q, S, A, R, t respectively refer to Q value, State, Action, Reward and timestamp).

$$Q_{new}(S_t, A_t) = Q_{old}(S_t, A_t) + \alpha\,[R_{t+1} + \nu * max\, Q(S_{t+1}, a) - Q(S_{t,} A_t)]$$

Equation 3

This equation illustrates the overview stracture of Q-learning, which is the process of obtaining the expected long-term rewards given the current state and the best actions within the restriction of greedy-policy.

4-2 Represent its performance

As an evaluation metric, average reward over 100 episodes is used to evaluate the model performance.

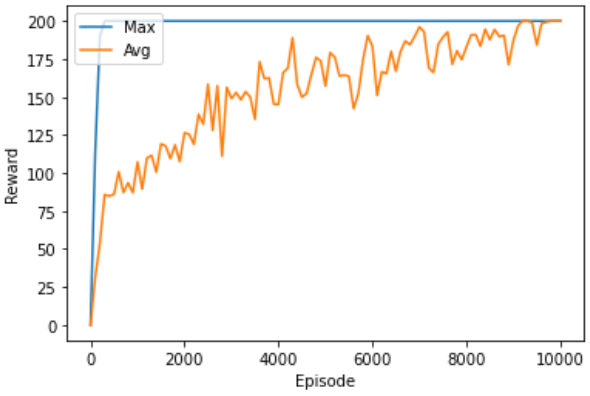

Figure 3

Figure 3 provides information about mean-reward and max-reward for each episode. Average reward increases sharply untill around 500 episodes until 80 mean-reward and gradually increases to the threshold (200) although the learning line has a tendency to be inverted to get to the maxium reward because of the exploration period.

5. Repeat the experiment with different parameter values, and policies

Moreover, in order to compare ther performance of hyper-parameters, several modifications have been specified for further in-depth investigation. The method used to explore the effect for hyper-parameters is called grid-search which allows the model to exclusively take every combination of parameters.

Learning rate	0.15	0.25	0.35
Gmma	0.5	0.8	0.99
Start-epcilon	0.2	0.5	0.8

Table 2

6. Analyze the results quantitatively and qualitatively

Figure 4 plots the average reward over the past 10 episodes for each of the 27 hyperparameter combinations. All hyperparameter configerations are colored based on the gamma value. 0.99 is green, 0.8 is blue and a 0.2 is red. It becomes clear that a high gamma is needed in order to solve the environment. Thus for the subsequent analysis we focus only on those hyperparameter configurations that have a gamma of 0.99.

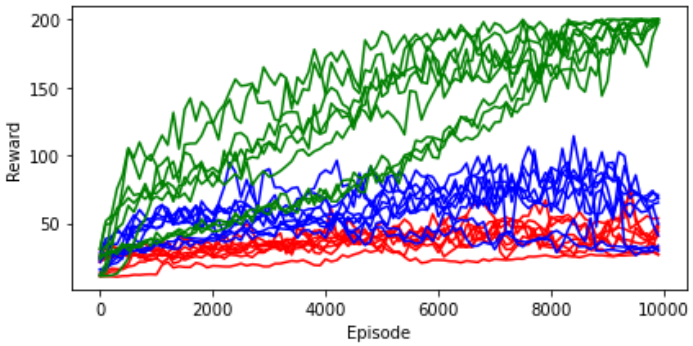

Figure 4

In figure 5, the average reward paths are colored based on the employed epsilon. We can observe that starting values of epsilon are able to solve the environment perfectly, a higher epsilon seems to reduce the variation in the rewards, by exploring a lot in the beginning and accepting lower rewards in earlier episodes. Furthermore, all tested values for the learning rate seem to be feasable choices, which do not materially affect the volatility or performance of the Q-learning algorithm if the appropriate policy is set.

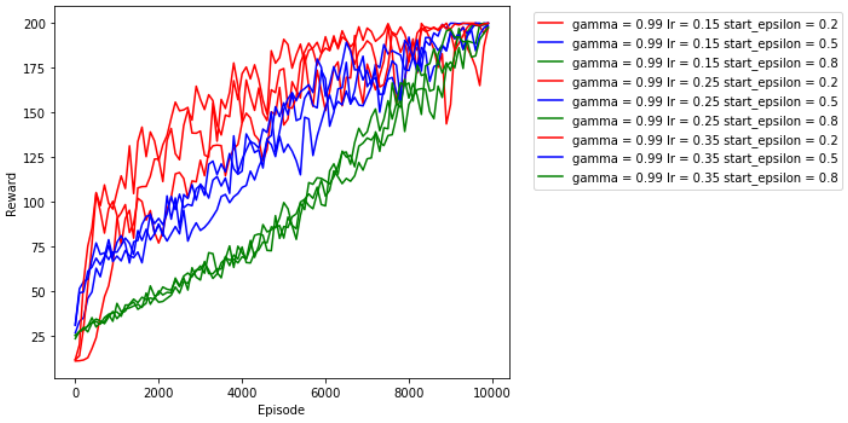

Figure 5

Advanced Task

7. Implement DQN with two improvements. Motivate your choice and your expectations for choosing a particular improvement.

7-1 Issues with current implementation and problem statement

As can be seen in the previous task, Q-learning performs well in a simple task to train an agent. However, due to the nature of high variance of gradient and convergence of the Q-network is slow, novel methods using deep learning have been placed along with the rapid development of neural networks which can arguably perform better than Q-learning. As an advanced task, in order to see the performane of DQN, Lunar-Lander from AI gym [1] will be examined as a case study. Figure 6 shows the example of the episode of Lunar-Lander. The task of Lunar-Lander is to successfully land the spacecraft by adjusting the fire main engine. Episode will finish if the lander crashes (-100) or lands in the area of landing-pad (+100) with the respective rewards. As an action space, four actions (do nothing, fire left orientation engine, fire right orientstion engine and fine main engine) is available in the descrete manner.

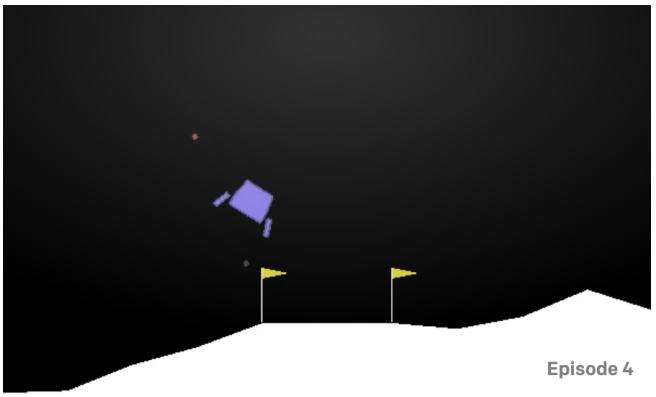

Figure 6 [1]

7-2 DQN

Although Q-learning is a simple and straightforward method, the limitation arises when Q-network is dealing with storing Q-values for computational reasons. DQN was first applied to Atari games by Deepmind in 2013 [5] to tackle those issues. In the original paper, the authors attemped to improve the model by adding experience reply buffers and neural networks. Experience replay buffers improve efficiency and stability by storing samples and DQN utlises neural networks to create the maps as opporsed to using Q-table.

7-3 Double DQN

Even though DQN has a strong benefit to improve the model capability, DQN tends to suffer from overestimation of rewards from random actions by chance. This might potentially harm the overall performance. In order to prevent this, Double DQN was introduced by Hado, Arthuer and David [6]. The central idea of Double DQN is to reduce the max operation in the target into the action selection and action evaluation resulting in more stable and robust development of the model [6]. As Double DQN only adds the same network by using the exsiting archtechture without adjusting new hyper-parameter and neural network, it seems to be true that Double DQN can be fairly easy to implement and yet has strong performance in comparison to the normal DQN. By implementing Double DQN, it is expected that the model will be less overoptimistic towards coincidence and more improvement can be archieved.

7-4 Dueling DQN

Dueling DQN was also proposed by Deepmind [7] as an extension of incoporating neural networks in reinforcement learning. The central idea of Dueling DQN is to reduce the inefficiency to learn the value of combination of state and action in each step by splitting into two streams.

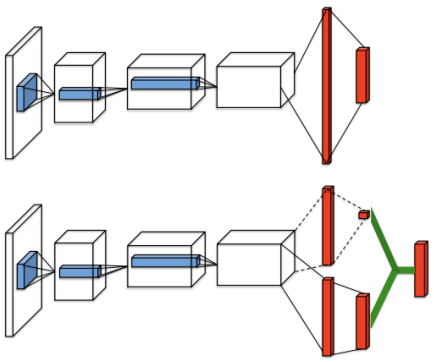

Figure 7 [7]

Figure 7 provides an overall architecture of DQN (on the top) and Dueling DQN (on the bottom). In Dueling DQN, instead of having a one-fully connected layer, two stream fully connected layers were deployed to process game-play simulated frameworks. One is used to estimate a state-value as a scalar and the other is used to estimate an advantage of each action as a vector. In the end, these two layers will be merged into one final output. By having this network, it allows the model to learn state-action functions in a more effective manner.

8. Analyse the results quantitatively and qualitatively

In ordoer to compare the results of the basline DQN to the two improvements, we focus on the rewards that these models achieve over the course of all training episodes. Here we especially focus on qualitative evaluation of the reward history, as well as quantitative measurements of performance, i.e., mean, median and standard deviation of the rewards.

In the first step, we train the baseline DQN model. We perform hyperparameter tuning to find the optimal parameters, however, due to a runtime of multiple hours, we only focus on learning rate and memory size. Figure 8 shows the training reward for all DQN hyperparameter combinations. To achieve a clearer view of the overarching trends inspite of high fluctuations in reward per episode, we apply a 200 episode moving average filter to the reward history.

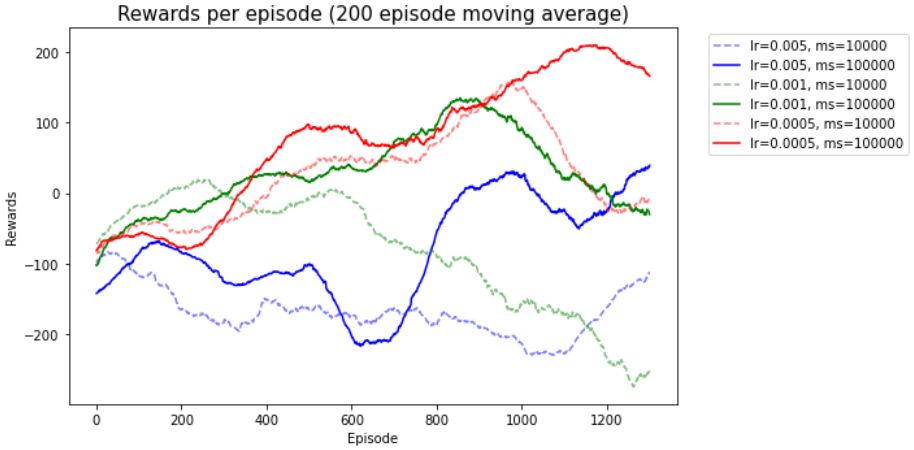

Figure 8

One can quickly see that for each learning rate, increasing the memory size by a factor of 10 leads to significantly better performance. The best performing model is the one with the lowest learning rate of 0.005 and the highest memory size of 100'000. Now, in a second step, these hyperparameters are used to train the double DQN and dueling DQN architecture. Figure 9 plots the moving average of rewards for the three network architectures.

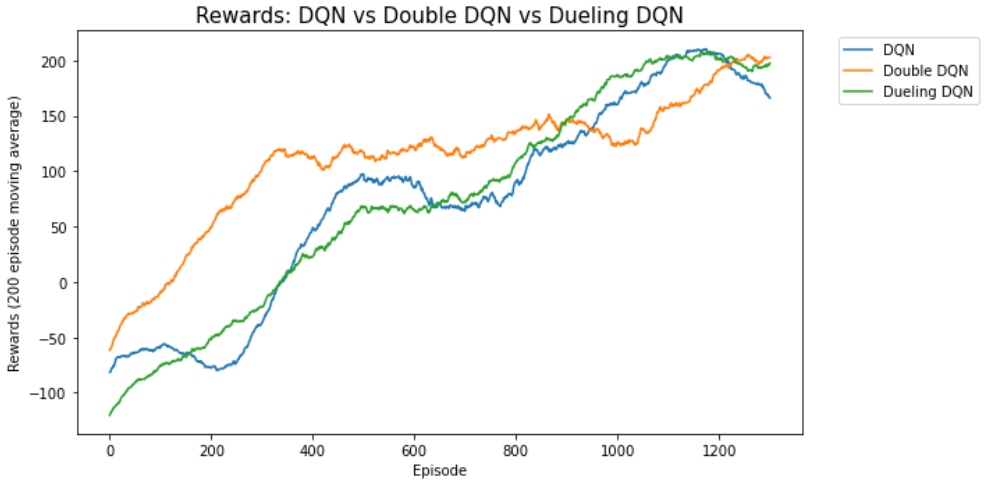

Figure 9

We see that for the first 800 episodes the double DQN has a higher average reward and seems to learn faster than the baseline and dueling DQN. However due to a longer plateau between episode 300 and 1100, the double DQN is the last to learn to solve the task consistently with an average reward of over 200. The dueling and baseline DQN have similar reward graphs. Although, the dueling DQN seems less volatile, whereas the DQN has multiple episodes where the average reward decreases significantly. This seems especially significant at the end of the training episodes, where the baseline DQN agent seems to forget the optimal solution, whereas dueling and double DQN architectures are able to continuously solve the environment. Figure 10 quantiatively confirms this intuition. It shows that the baseline DQN has the lowest average and median reward, while also having the highest standard deviation of rewards. The double DQN seems to be the best performing architecture with the highest mean and median reward and the lowest standard deviation of rewards.

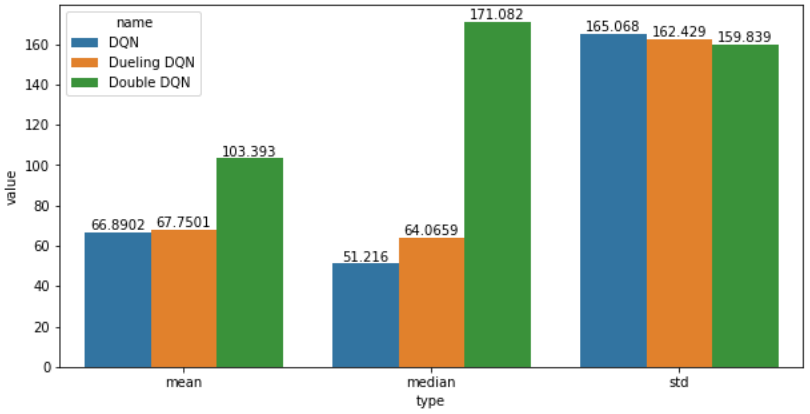

Figure 10

9. Apply the RL algorithm of your choice (from rllib) to one of the Atari Learning Environment. Briefly present the algorithm and justify our choice.

In this section, a more complex environment using MsPacman-v0, one of the Atari learning environments, will be applied to the training model shown by the example (Figure 11).

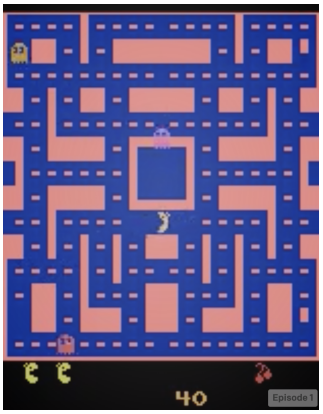

Figure 11 [1]

The main goal of this task is to train an agent which can maximise the score by eating as many pac-dots as possible whilist trying to escape from the ghost. Additionally, Pacman can eat the power pellet in the corner of

the screen, which can turn the ghosts blue or yellow and Pacman can get an extra point by eating the ghosts at that time. Reward used for this task is a game score. In order to train the algorithm, grid search for Dueling DQN and Double DQN will be applied. As it was mentioned earlier, Dueling DQN can reduce the inefficiency to learn the value and Double DQN can make the model less overoptimistic towards coincidence. Justification for using these algorithms is to observe how the model can perform in a much more complex environment in comparison to those previous environments. As a future reccomendation, more investigation for hyper-parameters will be necessary as there is a computational limitation on the environment. However, this case study will suffice the expectation for the experimental motivation.

10. Analyse the results quantitatively and qualitatively.

Figure 12 shows the best hyper-parameter for the model in line with the evaluation metrics (mean reward for each episode). The best parameter for this training during 100 episodes is gamma: 0.9, learning rate 0.005 with Double DQN. During the very beginning of the episode, it is true that a more drastical increase of mean rewards can be observed. One of the reasons and hypothesis for this trend could be explained by the way to get a reward in a Pacman game. At the beginning, agents can get a score by moving around the cell and eating pac-dots. However, in order to get more score, Pacman has to strategise the route to eat the power pellet as well as the way to escape from the ghosts. Hence, after some threshold (in this case, around 20 episodes), it seems to be true that the increasing rate will be slower compared with the beginning because of the agent's strategy.

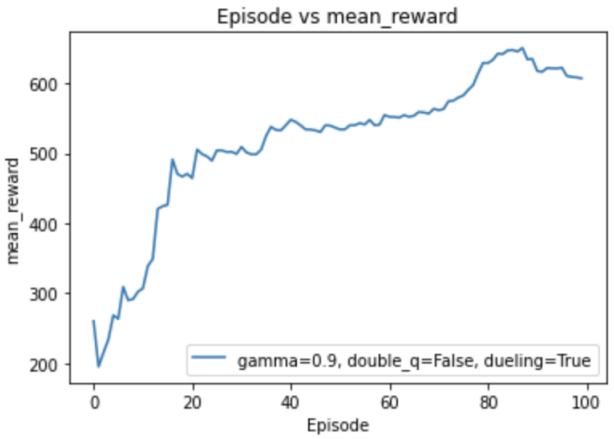

Figure 12

In addition, although there is an overall increase in the reward in line with the number of episodes, some ups and downs can be seen during the training. Potential reasons for this could be explained by exploitation vs exploration. Taking a greedy epsilon approach can allow one to look for future rewards rather than falling down the pitfall of immidiate reward. Therefore, although some episodes have a lower score in the later episodes, this might contribute to the future rewards.

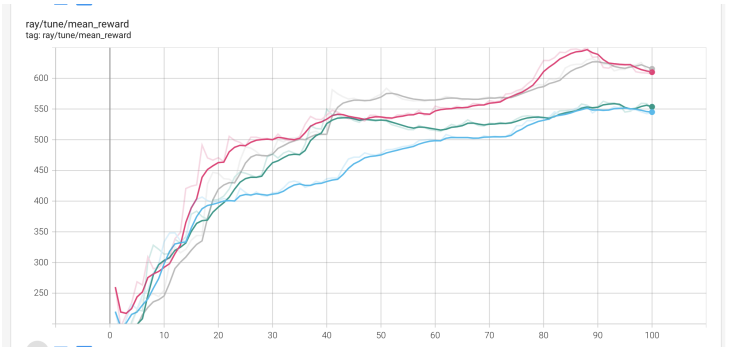

Figure 13

Figure 13 shows a visualisation of comparison of using DQN, Double DQN and Dueling by using a tensorboard. The learning curve for Blue line (Vannila DQN) seems to be slower than other models from the beginning to the end of episodes whereas Green Line (Double DQN) has better scoreing metrics at the beginning and converges to the low rewards at the end of episode. Moreover, Red Line (Dueling DQN) and Gray line (Double DQN and Dueling DQN) seem to have a high performance in this case study. This tendency might indicate the trend for taking advantage of using Dueling DQN which enables one to have better learning speed by separating state-value and an advantage for action. However, future examination using more parameters and powerful computational ability will be needed to investigate the factors contributing to the rewards.

11. Implementation of PPO

PPO (Proximal Policy Optimisation) is another novel method to contribute to the great performance in reinforcement learning. The motivation for this algorithm is to solve the issues of achieving good results using the 'policy gradient method by avoiding many policy updates. Furtheremore, as sample efficiency is very low, it may take a considerable amount of steps just to learn a simple task. Even though some algorithms such as TRPO (trust region policy optimisation) were invented to tackle this, the implementation of TRPO has been complicated and does not have good compatibility for the architechture including noise or parameter sharing [8]. In order to improve this, PPO can enhance the efficiency by clipping to avoid big changes in policy. In this case study, we will also try to implement PPO in Lunar-Lander. We also conducted grid-search to tune the learning rate (0.1, 0.001, 0.0001 respectively).

Episode Reward Mean	Episode Reward Max	Learning Rate
-132.068	26.204	0.01
200.376	290.861	0.001
282.199	317.006	0.0001

Table 3

Table 3 provides an overall result for PPO applying to Lunar Lander. Although decreasing the learning rate such as 0.0001 takes a huge amount of time, it seems to be true that we can obtain much better scores in comparison to other learning rates. Given that running time for 2000 iterations with only 3 parameters takes

approximiately 15 hours (53607 seconds) with GPU, the trade-off relathionship between computational ability and model performane should be taken into further consideration. Having said that, the overall result for best learning rate seems to be significantly better than the previous model such as DQN.

References

[1] OpenAI environments. OpenAI. [online]. Available at: https://gym.openai.com/envs

[2] Barto, A., Sutton, R. and Anderson, C., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5), pp.834-846.

[3] Bulut, V., 2022. Optimal path planning method based on epsilon-greedy Q-learning algorithm. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 44(3).

[4] Jang, B., Kim, M., Harerimana, G. and Kim, J., 2019. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7, pp.133653-133667.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. 2013, "Playing Atari with Deep Reinforcement Learning".

[6] Van Hasselt, H., Guez, A. & Silver, D. 2015, "Deep Reinforcement Learning with Double Q-learning".

[7] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. and de Freitas, N., 2022. *Dueling Network Architectures for Deep Reinforcement Learning*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1511.06581> [Accessed 22 April 2022].

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2022. *Proximal Policy Optimization Algorithms*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1707.06347> [Accessed 23 April 2022].

The contribution of individuals to the team task with contribution ratio

Basic Task

- **Problem setting and background reserach** (Yuichi 60% Setfan 40%)

Yuichi did problem setting and academic research to find relevant academic sources.

Stefan actively gave Yuichi feedback about how to deepen our understanding of problem tasks and how we need to implement our code by zoom discussion.

- **Implementation including coding and visualisation (**Yuichi 40% Stefan 60%**)**

Yuichi helped Stefan's baseline code by having live feedback (including editing and modification) and finding related external resources.

Stefan wrote a code as a baseline based upon the references and implemented visualisation.

- **Report (**Yuichi 50% Stefan 50%)

Yuichi mainly wrote a draft version of the report based upon the model.

Stefan gave constructive detailed feedback to estibalish our documents and implication.

Advanced Task

- **Problem setting and background reserach (**Yuichi 60% Stefam 40%**)**

Yuichi helped find an appropriate problem setting and did some academic reserach to implement DQN and the two other improvements.

Setfan gave detailed feedback based upon Yuichi's findings and helped establish our initial research.

- **Implementation including DQN, DDQN, Dueling** (Yuichi 40%, Stefan 60%)

Based upon the background research, Yuichi helped with an implementation of Double DQN and Dueling DQN and gave constructive feedback about how to present our model including HP tuning and visualisation along with debugging the code.

Stefan wrote the baseline of DQN code based on the lab material and adapted to our model specification. Also, Stefan visualised the model interpretation and did computation using his own GPU.

- **Report (**Yuichi 50% Stefan 50%**)**

Yuichi mainly wrote a draft version of the report based upon the model.

Stefan gave constructive detailed feedback to establish our documents and implication.

- **PPO: Background reserach and Implementation (**Yuichi 50% Stefan 50%**)**

Yuichi wrote the draft version of coding and did background reserach.

Stefan debugged our code and gave feedback for the room of improvement to Yuichi.

- **PPO: Report** (Yuichi 50% Stefan 50%)

Yuichi wrote the draft version of the report based upon the model result.

Stefan reviewed the report and enriched the content by doing additional academic reserach.

Overall. The workload of team tasks has been equally split and Yuichi and Stefan engaged with every part of tasks by having a constant 1 on 1 meeting and code debugging session.