MTech(IS)

Breakout Reinforcement Learning System

Project Report

**REINFORCEMENT LEARNING**

**Team Members**

CAO LIANG – A0012884E

GENG LIANGYU – A0195278M

HAN DONGCHOU FRANCIS – A0195414A

ONG BOON PING – A0195172B

TAN CHIN GEE – A0195296M

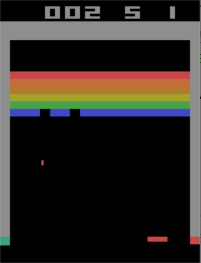
1.0 EXECUTIVE SUMMARY

The Breakout RL (reinforcement learning) system is a system that learns to play the Atari Breakout game with reinforcement learning. It exhibits the techniques and strategies for resolving the general MDP (Markov Decision Processes) problems and can be applied to real world similar problems.

We considered and chose the Breakout game since it contains all the basic features of MDP problem, and also allow us to gradually improve our learnings within the tight project schedule.

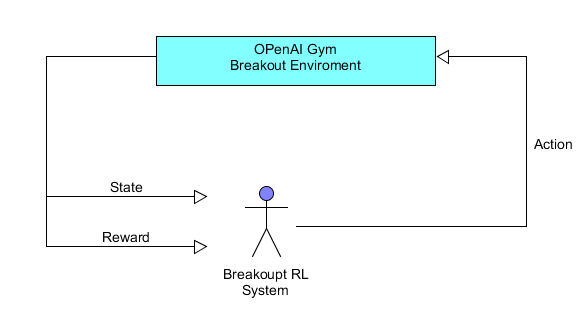
2.0 PROBLEM DESCRIPTION

The Breakout game contains eight rows of bricks, and the player will move the paddle to hit the ball. If the ball knocks down the bricks, the player will gain the marks. If the player missed 3 times to hit the ball by moving the paddle, then the player loses the turn, and new turn will begin which resets the player’s existing marks. The higher the player’s marks, the better the player plays.



**Fig 2-1 Breakout Game**

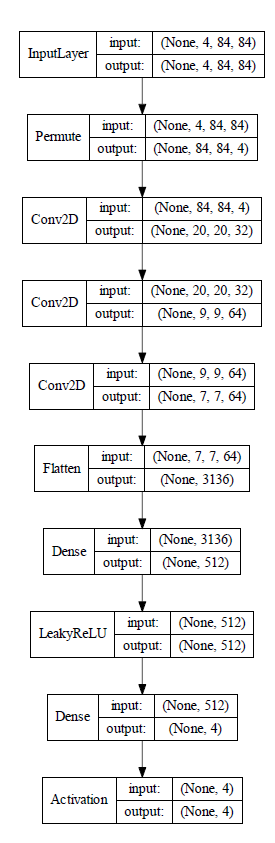
The OpenAI Gym system simulates the Breakout environments, and we decide to select “BreakoutDeterministic-v0” environment for our system. The Breakout RL system is required to play OpenAI Gym system breakout game to achieve highest marks as it can. The Breakout game problem is abstracted and illustrated in Fig 2-2.



**Fig 2-2 Breakout Game Problem**

3.0 SOLUTION

The DQN (Deep Q-learning Network) is designed for Breakout RL system to address the issue. The DQN structure is shown in Fig 3-1.



**Fig 3-1 DQN Network Structure**

**3.2 System Design/Model – Components of the System**

The Breakout RL system is divided into 3 parts: DQN network model, DQN Agent, and Breakout Game environment. The design is illustrated below (Fig 3-2).

**Fig 3-2 System Design**

The DQN network model consists of the core of the solution design which in charge of learning the rules from the Breakout game, and also utilizing the rules to play the game.

The Breakout game environment is the pre-built game environment at OpenAI Gym system (<https://gym.openai.com/>), and it will simulate the real Breakout game environment.

The DQN Agent is pre-built class at keras-rl package (<https://github.com/keras-rl/keras-rl>), and it connects the DQN network and Breakout Game, so it receives the DQN network actions and sends them to Breakout game environment, also it transfers back the reward and new state of game to the DQN network.

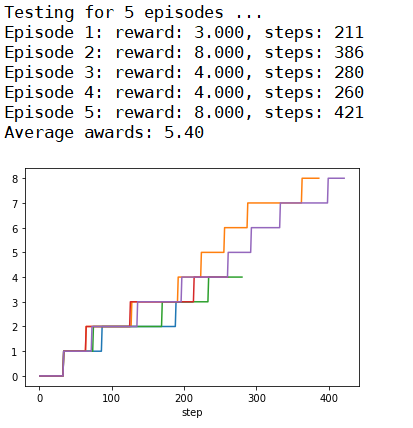
4.0 RESULTS

We have managed to use reinforcement learning to make a classic Atari game plays on its own, without a human player. After training for 500, 000 steps, the Breakout RL System was able to play the game with an average reward marks of 5.40 as shown in Fig 4-1.

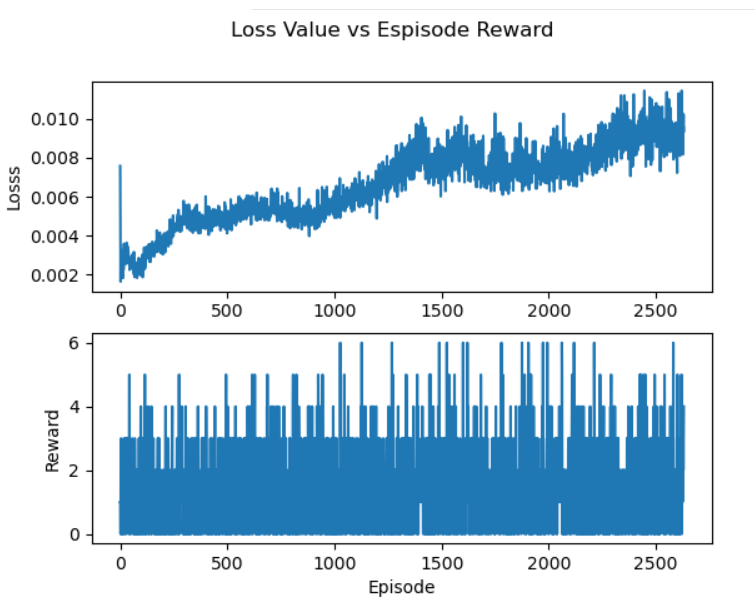
**4.1 Findings**

4.1.1 Kernel Initialization Parameter

We explicitly specified “HE” normalization for the kernel initalization that follow the best practice. However, the training results did not improved much as shown in Fig 4-2. The game rewards are below 6 during training.



**Fig 4-1 Average Rewards Result**



**Fig 4-2 Kernel Initialization He Normalization**

4.1.2 DQN Agent Target Model Update

This controls how often the DQN network is updated. We have changed its value from 0.01 to 10,000 to follow the Google DeepMind code based on the blog (<https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756>).

4.1.3 Policy

The policy definition code is shown below, and we have learned that the policy makes the most significant differences on the system results. We have tested for Annealing Epsilon Greedy Policy and Boltzmann Q Policy, and the training results are shown in Fig 4-3 and Fig 4-4.

policy = LinearAnnealedPolicy(EpsGreedyQPolicy(),

attr='eps',

value\_max=1.,

value\_min=.1,

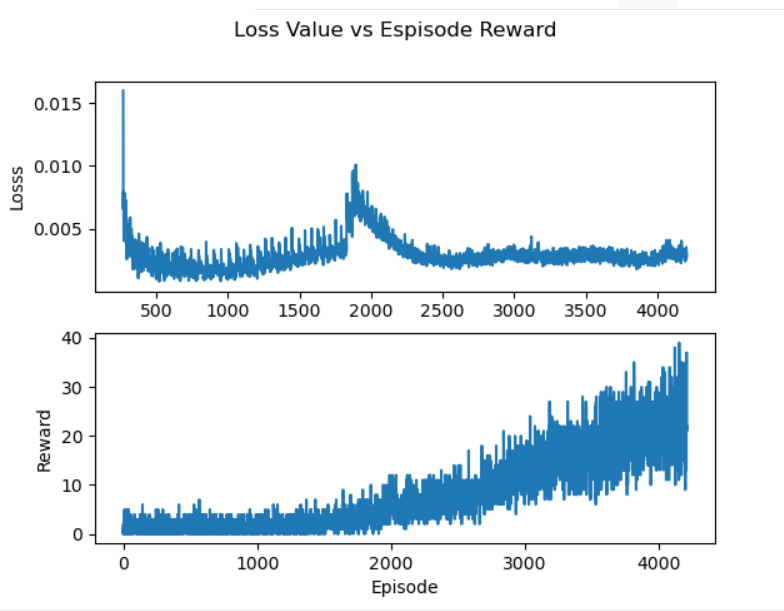
value\_test=.05,

nb\_steps=1000000)

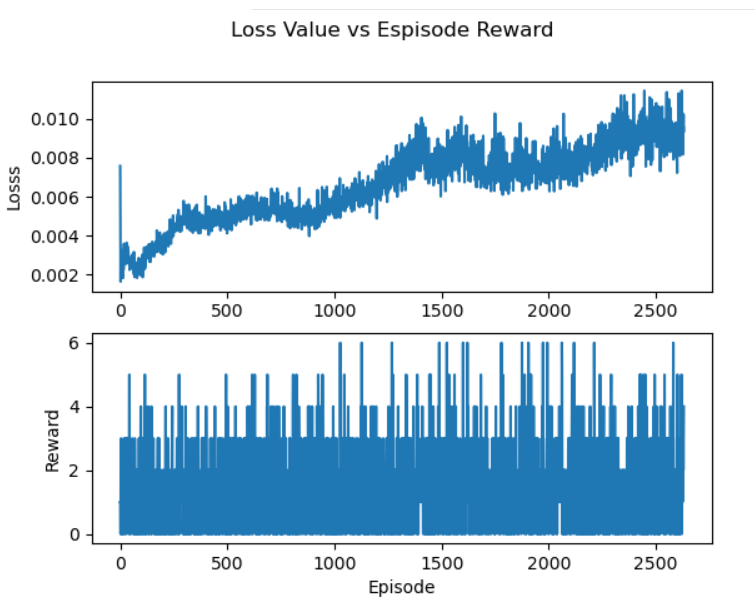
The Epsilon Greedy Policy selects the action with highest Q value and only selects random action with a specfied probability given by “eps”. There is no change in the value of “eps” during training. On the other hand, the annealing policy allows the control of the values of “eps” from 1.0 to 0.1 in 1 million steps. Therefore, the action taken at the beginning of training will include more exploration (high value of “eps”), and gradually the action will move toward exploitation by following actions with the highest Q value (low value of “eps”).

The Boltzmann Q Policy chooses its action based on a weighted probabilities of all actions’ estimated Q-values, instead of the binary distinction that Epsilon Greedy Policy employs to guide its selection of action. For example, Epsilon Greedy Policy will treat the second best action as having the same magnitude as the least optimal action, and therefore not consider this action for selection. But Boltzmann Q Policy would distinguised between the second best action and the least optimal action for its selection, which would help it to focus attention on more promising but least optimal actions.

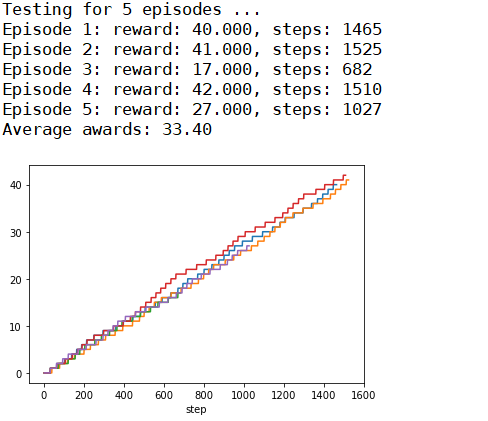
Since the enviroment “BreakoutDeterministic-v0” repeats the previous action 25% of the time in the current step, the Annealing Epsilon Greedy Policy is more suitable for acheiving better results. And the test results proved the conclusion as shown in Fig 4-5 and Fig 4-6. The average rewards reached 33.40 with Annealing Epsilon Greedy Policy, and are 519% higher than Boltzmann Q Policy’s reward of 5.40.



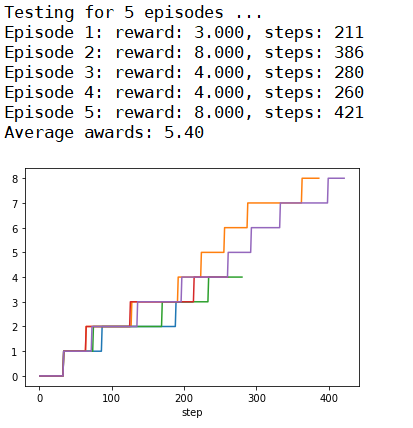
**Fig 4-3 Annealing Epsilon Greedy Policy**



**Fig 4-4 Boltzmann Q Policy**



**Fig 4-5 Annealing Epsilon Greedy Policy Test Results**

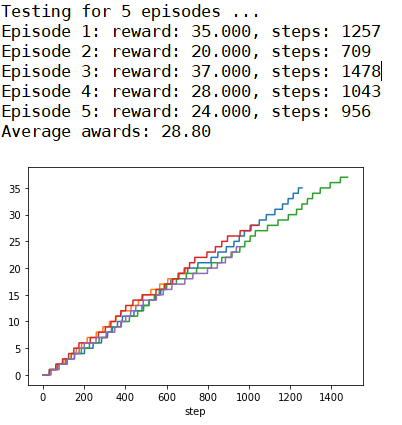


**Fig 4-6 Boltzmann Q Policy Test Results**

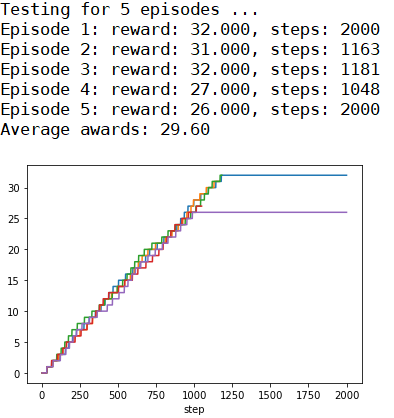
4.1.4 Training Steps

We first conducted training from scratch on Breakout RL system using 1,700,000 steps, and then loaded the training weights to re-train the Breakout RL system again using 1,700,000 steps.

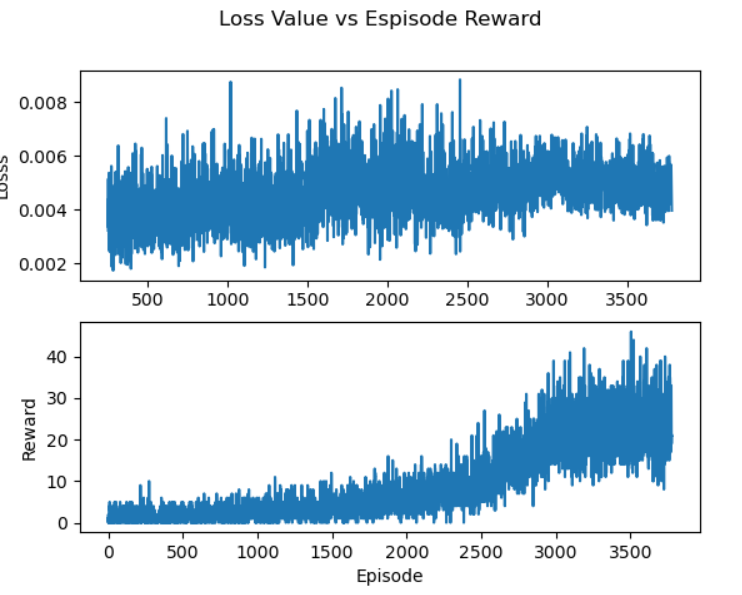
The test results for the initial training and re-training are shown in Fig 4-7 and Fig 4-8. It shows that re-training rewards (28.80) had improved 2.7 % compared with the initial training (29.60). The re-training status plotted in Fig 4-9 shows that the episode rewards slowly increased at very low marks (below 10) from the beginning to 2,000 episodes, and reached 20 marks after 2,500 episodes. After 3,000 episodes, the episode rewards of 30 or higher were reached. The rewards increased from low to high marks slowly due to the buiding of the internal Q value table, and so in order to achieve good results, we recommend that the training steps to be at least 1,700,000 steps.



**Fig 4-7 Initial Training Test Results**



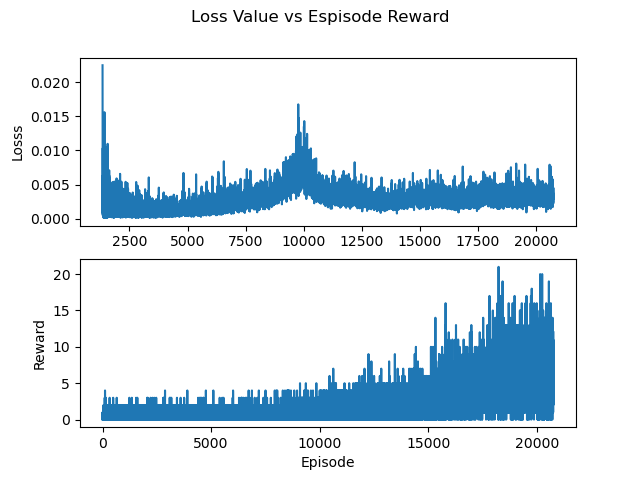
**Fig 4-8 Re-training Test Results**



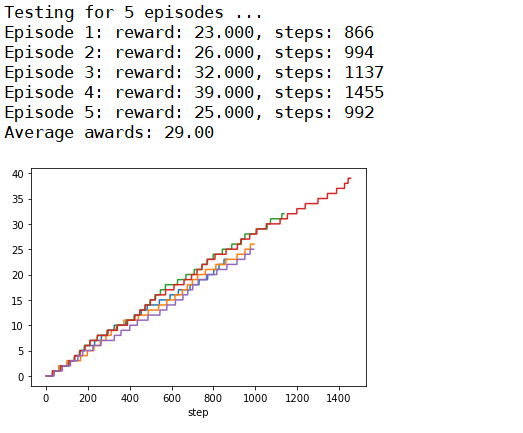
**Fig 4-9 Re-training Status**

4.1.5 Game Life Lost

We have read the article (<https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756>). It describes that passing the terminal state to the replay memory when a player turn is lost will increase the game rewards significantly. So, we have modified keras-rl package “core.py” to detect game life lost and start new episode when this game life lost has happens. The training and testing results are shown in Fig 4-10 and Fig 4-11. It shows that the improvement of the rewards is much lower 0.69% (from 28.80 to 29.00). Therefore, there may be better ways to utilize the detection of game life besides restarting episode. Due to time constrains, we have not explored better ways of doing so.



**Fig 4-10 Training Results for Game Life Lost**

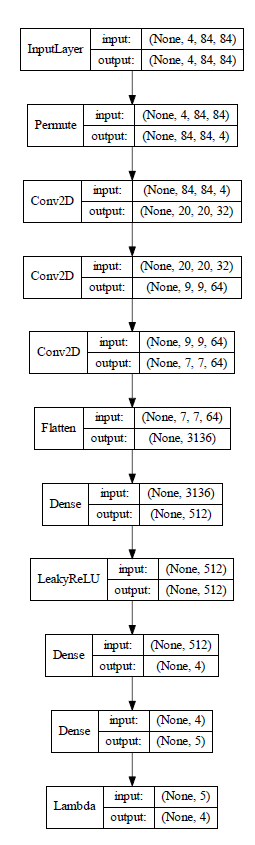


**Fig 4-11 Testing Results for Game Life Lost**

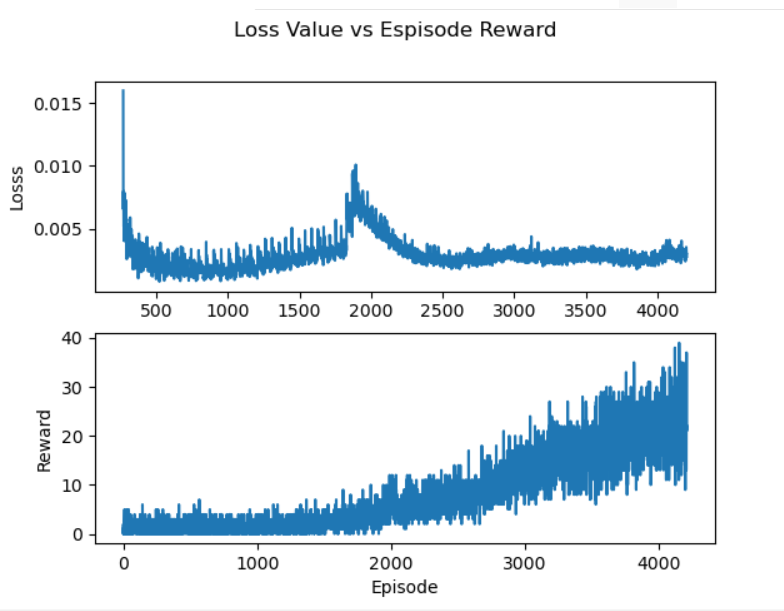
4.1.6 Dueling Network

The dueling networks are pre-built option at DQN keras-rl package, and it allows the dueling network layers to be automatcally added to the existing model. The dueling network structure of the Breakout RL system is shown in Fig 4-12. The training and testing results are shown in Fig 4-13 and Fig 4-14.

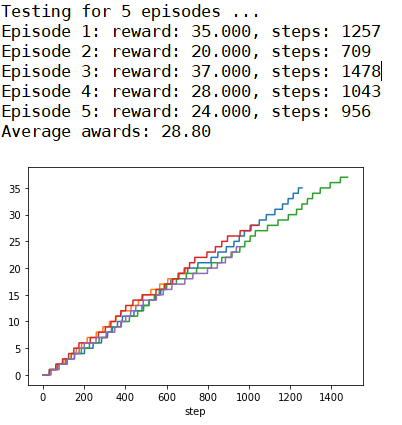
The test results without dueling network is shown at Fig 4-15. Comparing the two networks, the average rewards without dueling network (33.40) are 15% higher than that of the dueling network (28.80). Therefore, the dueling network does not help in obtaining better rewards from this experiment.



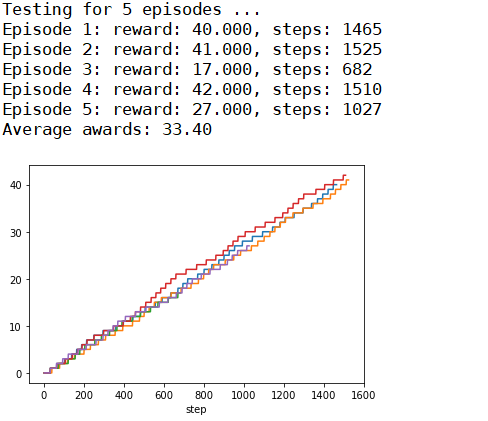
**Fig 4-12 Dueling Network Model Structure**



**Fig 4-13 Dueling Network Training Results**



**Fig 4-14 Dueling Network Test Results**



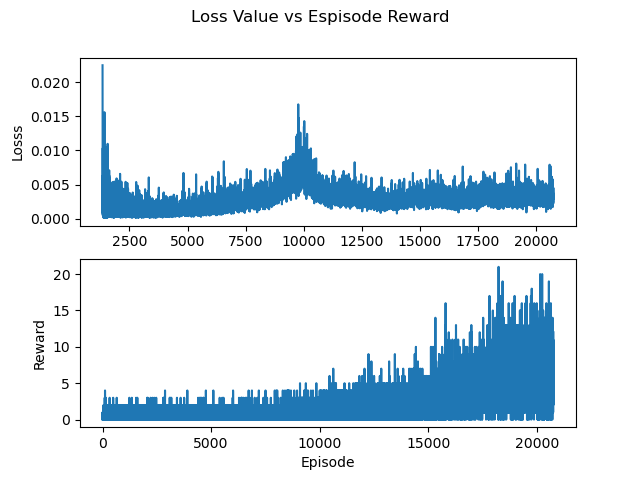
**Fig 4-15 Test Results Without Dueling Network**

4.1.7 Different Environment

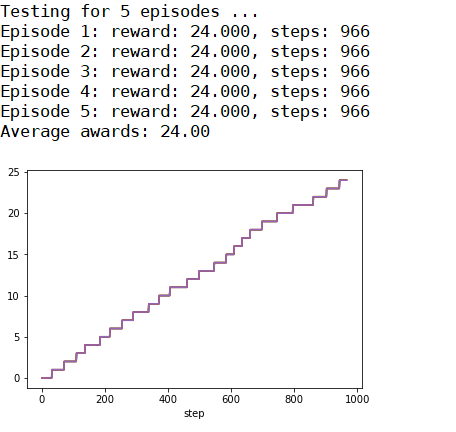
We have also tested the “BreakoutDeterministic-v4” environment and compared it with the current environment “BreakoutDeterministic-v0”.

The “BreakoutDeterministic-v0” repeat\_action\_probability is 0.25, and so 25% of the time the previous action will be repeated. The “BreakoutDeterministic-v4” repeat\_action\_probability is 0, and so it will never repeat a previous action.

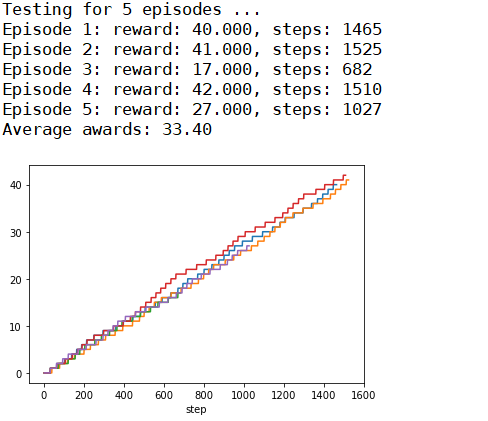
The training and test results for “BreakoutDeterministic-v4” are shown in Fig 4-16 and Fig 4-17. Compared with “BreakoutDeterministic-v0” test results in Fig 4-18, it shows that “BreakoutDeterministic-v0” has achieved better rewards (33.40) than “BreakoutDeterministic-v4” (22.40) using the same policy (Annealing Epsilon Greedy Policy). This was due to the uncertainty of actions in “BreakoutDeterministic-v0” that were better handled with Annealing Epsilon Greedy Policy. So if the RL system environment is changed, the RL system policy should also be tested and adjusted in order to get better results.



**Fig 4-16 BreakoutDeterministic-v4 Training Results**



**Fig 4-17 BreakoutDeterministic-v4 Test Results**



**Fig 4-18 BreakoutDeterministic-v0 Test Results**

5.0 CONCLUSION

The Breakout RL system has proved to be able to learn playing the game with DQN network. Even though its reward marks were not high, it has showed that the DQN system was able to learn to play the game itself by learning from the game screen images and interacting with the game environment. Therefore, it provides the practical solution to similar real-world problem, and shows the power of reinforcement learning.

**5.1 Observations & Insights**

A key insight that we had observed was that, given the same number of training steps, test results was better when we trained the system at one go, instead of breaking up the training in two parts and applying the training weights from the initial training to the re-training.

We have not implemented some measures to improve the Breakout RL system results. E.g. changing the keras-cl package to implement better ways to utilize the detection of game life lost such as changing the process memory status, or using a GPU to reduce training time.

Also, we did not train the RL system for longer steps such as 3,000,000 steps or 4,000,000 steps due to the assignment time constrains. The longer steps can lead to better results. Therefore, there still leaves much improvement space for the Breakout RL system to achieve higher reward marks.

APPENDIX A. USER MANUAL

1. Setup Guide

* Install keras-applications and tensorflow

pip install keras-applications==1.0.7 tensorflow==1.13.1

* Install tensorflow-gpu for GUP training (Optional)

pip install tensorflow-gpu==1.13.1

* Install gym by OpenAI

pip install gym

* Install h5py

pip install h5py

* Install Pillow:

pip install Pillow

* Install gym[atari]: Atari module for gym.

pip install gym[atari]

* Install WandbLogger

pip install wandb

* Install pandas

pip install pandas

1. Execution

* Training at BreakoutDeterministic-v0

python sls\_breakout.py train v0

* Show Training Results at BreakoutDeterministic-v0

python sls\_breakout.py plot-train v0

* Testing at BreakoutDeterministic-v0

python sls\_breakout.py test v0

* Training at BreakoutDeterministic-v4

python sls\_breakout.py train v4

* Show Training Results at BreakoutDeterministic-v4

python sls\_breakout.py plot-train v4

* Testing at BreakoutDeterministic-v4

python sls\_breakout.py test v4

APPENDIX B. CODES & DATASETS

File: sls\_breakout.py

* Main application for Breakout RL system

File: dqn\_BreakoutDeterministic-v0\_weights.h5f

* Weights records for DQN network at BreakoutDeterministic-v0

File: dqn\_BreakoutDeterministic-v4\_weights.h5f

* Weights records for DQN network at BreakoutDeterministic-v4

File: Readme.md

* Readme file for execution setup

Folders: rl, utils

* Keras-rl packages files

Folder: history

* Training records