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

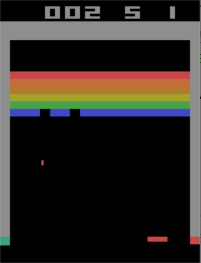
1.0 EXECUTIVE SUMMARY

The Breakout RL (reinforcement learning) system is the system which learns to play 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 consider to choose the Breakout game since it contains all the basic features of MDP problem, and also allow us to gradually improve within the tight project schedule.

2.0 PROBLEM DESCRIPTION

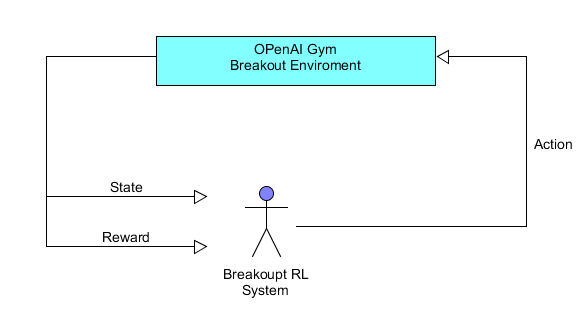
The Breakout game contains eight rows of bricks, and the player will move the paddle to hit the ball. If the ball knocks down the bricks, the player will gain the marks. If the player missed 3 times to hit the ball by moving paddle, then the player loses the turn, and new turn will begin which resets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

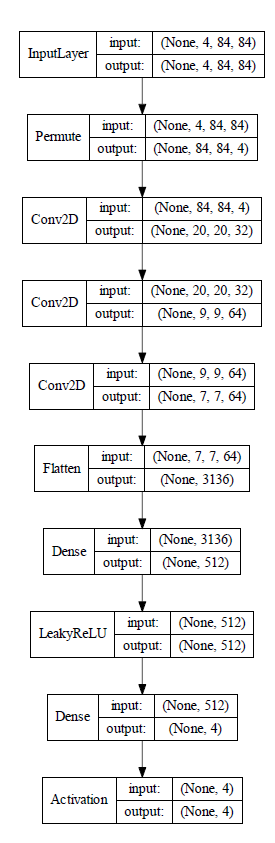


**Fig 2-2 Breakout Game Problem**

The Breakout RL System choses the action and send to the OpenAI Gym Breakout environment, then the Breakout environment returns back the new state of the Breakout game, and the reward for the action taken. The Breakout RL system is required to achieve highest reword (marks) as it can.

3.0 SOLUTION

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



**Fig 3-1 DQN Network Structure**

**3.1 Reasons for Why DQN is Suitable to Solve this Problem**

The DQN adopts Q-Learning method to approximate the breakout game state-action pairs Q-function from the interaction with the breakout environment. It will build a table of Q values, Q(s, a) that represents the expected

reward of Breakout game action “a” at the Breakout environment state “s”. Then it improves on this table by interacting with the Breakout environment until the optimal Q table is found. The optimal Q table provides the optimal policy to play the game and it is improved by learning with more data using deep neural network.

The deep learning neural network is proven to be universal function approximator, and also it is performed extremely good on image processing. Therefore, the DQN adopts the deep learning neural network to build the optimal Q-learning solution.

The conv2D layers in DQN network are used to capture the features of the game screen images since the inputs of the DQN network are the 4 sequential game screen images. The 4 sequential game images represent the game action direction and flow, so it avoids the issue which will be caused by 1 game screen image.

The last dense layer in DQN network outputs vector values which represent all the available actions at the Breakout game. The maximum value within the final output values indicates the corresponding action should be the best action to taken based on DQN network.

**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 System Design**

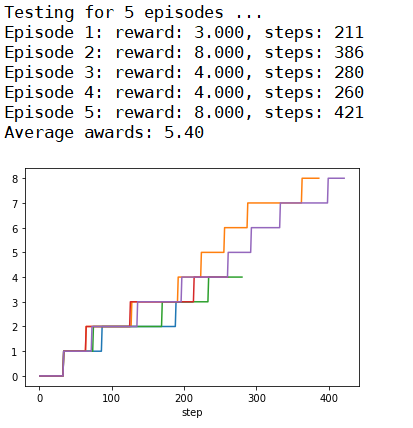
The DQN network model consists of the core of the solution design which in charge of learning the rules from the Breakout game, and also utilizing the rules to play the game.

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

After training for 500, 000 steps, the Breakout RL System was able to play the game with average reward marks 5.40 as shown in Fig 4-1.

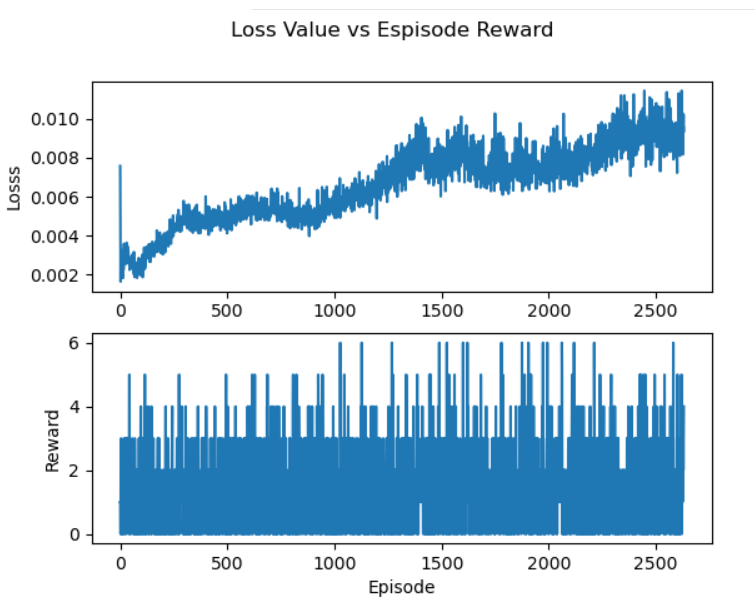


**Fig 4-1 Average Rewards Result**

**4.1 Findings**

4.1.1 Kernel Initialization Parameter

The kernel initalization is explicitly specified to “HE” normalization to follow the best practice, and the training results do not improve much as shown in Fig 4-2. The game rewards are below 6 during training.



**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 10000 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 resutls are shown in Fig 4-3 and Fig 4-4.

policy = LinearAnnealedPolicy(EpsGreedyQPolicy(),

attr='eps',

value\_max=1.,

value\_min=.1,

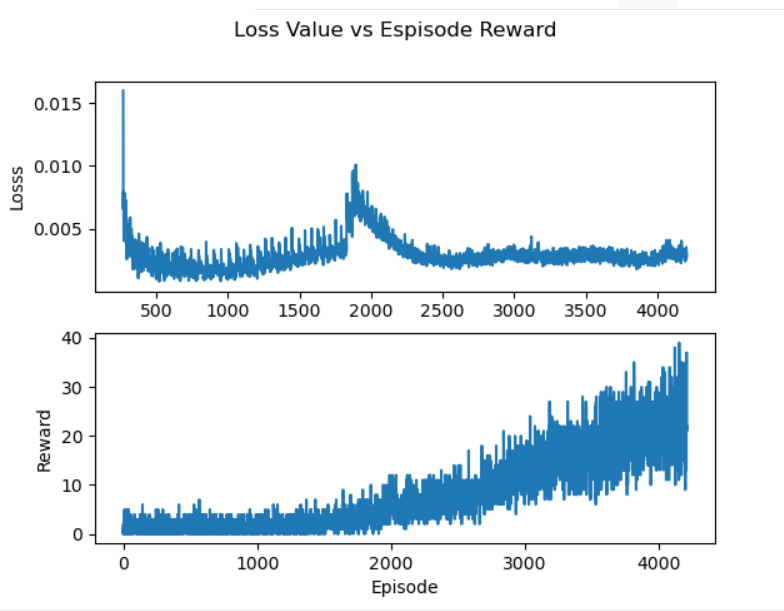
value\_test=.05,

nb\_steps=1000000)

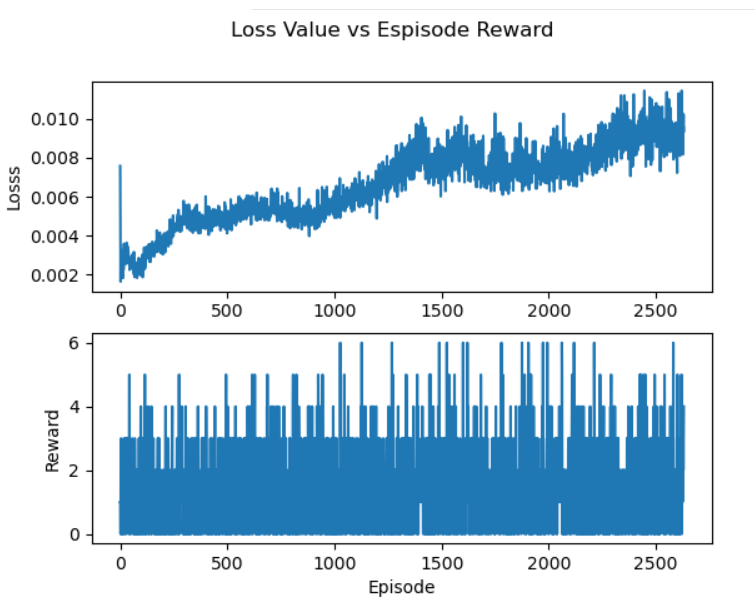
The Epsilon Greedy policy selects the action with highest Q value and only select random action in certain probablilty as “eps”. The anneling policy controls the value of “eps” from 1.0 to 0.1 in 1 million steps. So the action taken at the beginning towards the exploration (high value of “eps”), and gradually the action will toward the exploitation and fllow the Q value (low value of “eps”).

The Boltzmann Q Policy determins Q value based on the probabilty, and selects the action randomly based on Q value. It towards to exploration all the ways from the beginning to the end.

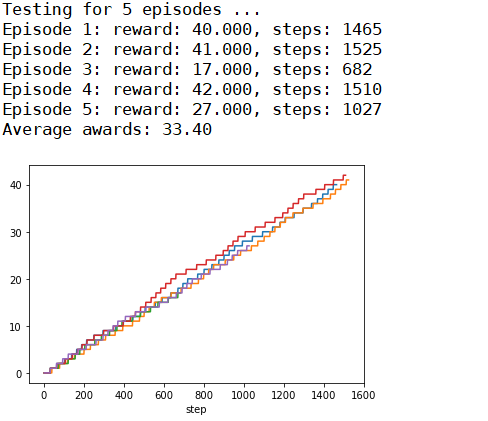
Since the enviroment “BreakoutDeterministic-v0” repeats the same action 25% of the time as the previous action, the Annealing Epsilon Greedy policy is more suitable for achiving better results. And the test resuls prove the conclusion as shown in Fig 4-5 and Fig 4-6. The average rewards reach 33.40 with Annealing Epsilon Greedy, and are 519% higher than Boltzmann Q Policy (5.40).



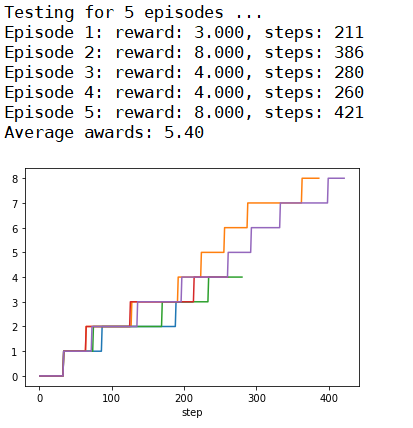
**Fig 4-3 Annealing Epsilon Greedy Policy**



**Fig 4-4 Boltzmann Q Policy**



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

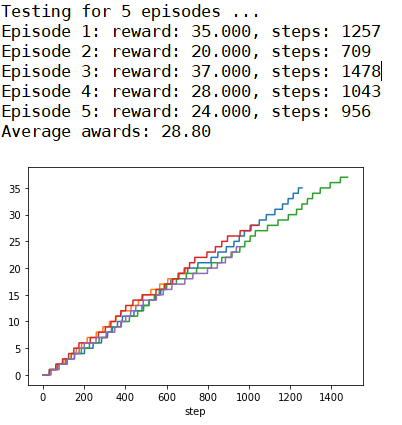


**Fig 4-6 Boltzmann Q Policy Test Results**

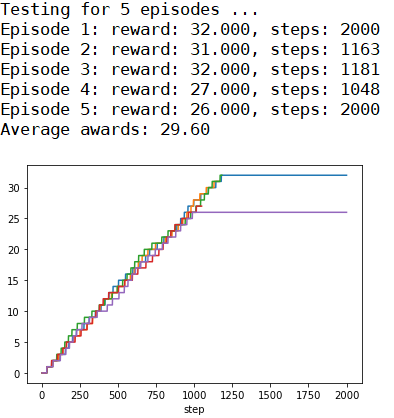
4.1.4 Training Steps

We first conducted training from scatch on Breakout RL system in 1,700,000 steps, and then loaded the weights just trained, and re-trained the Breakout RL system again in 1,700,000 steps.

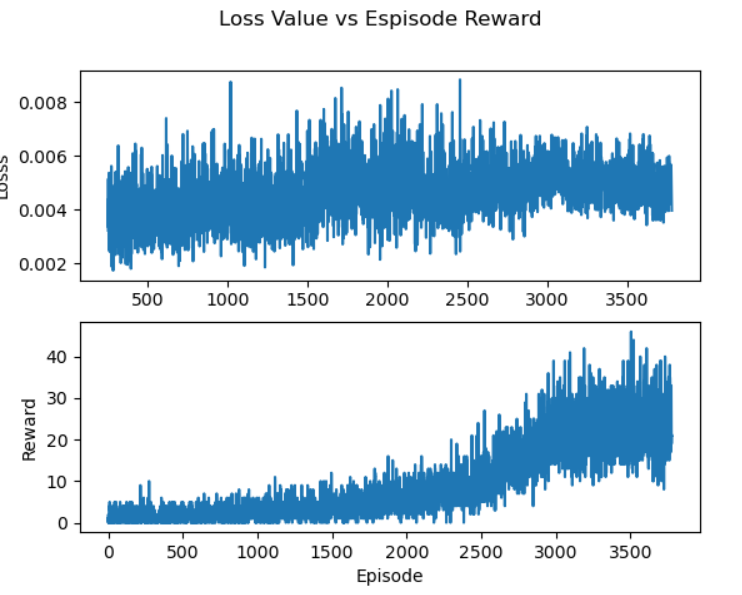
The test results for fresh training and re-training are shown at Fig 4-7. It shows that re-training rewards (28.80) imporve 2.7 % than fresh training (29.60). The re-training status at Fig 4-9 shows that the episode rewards are slowly increased at very low marks (below 10) from the beginning to 2000 episodes, and reached 20 after 2500 episodes. After 3000 episodes, the rewards 30 or above are reached. The rewards are increased from the low to high slowly due to the buiding of the internal Q value table, so in order to achive good results, the training steps will be recommended to be at 1,700,000 steps.



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



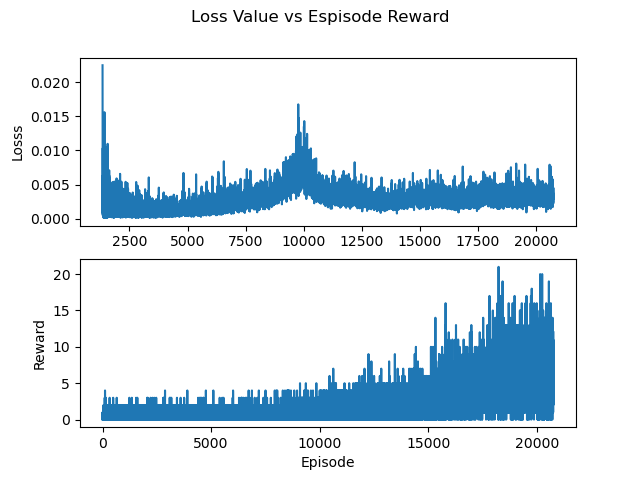
**Fig 4-8 Re-training Test Results**



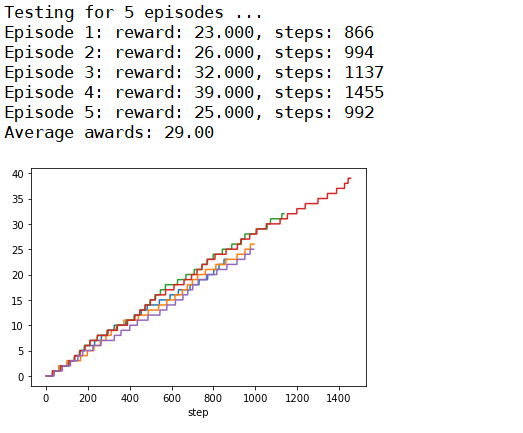
**Fig 4-9 Re-training Status**

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 happens. The training and testing results are shown at 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 for better ways.



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

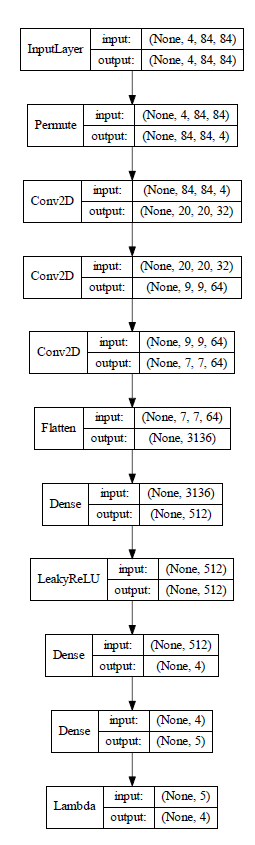


**Fig 4-11 Testing Results for Game Life Lost**

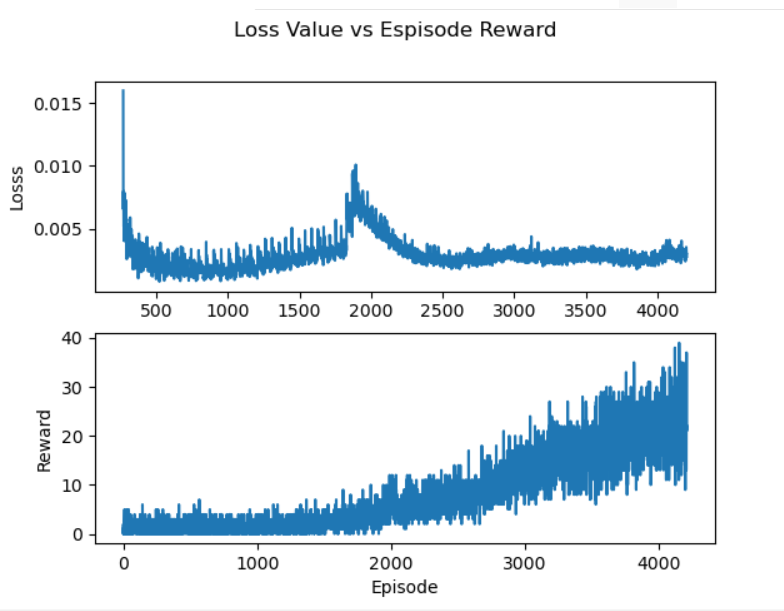
4.1.5 Dueling Network

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.

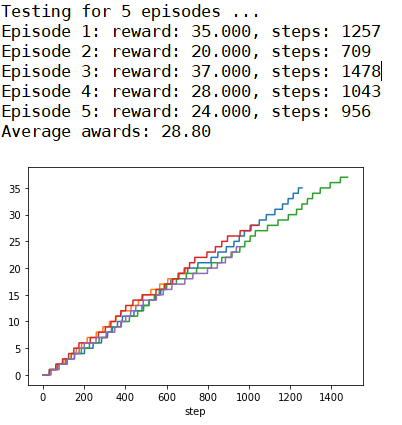
The test results without dueling network is shown at Fig 4-15. Compared to the two networks, the average rewards without dueling network (33.40) are 15% higher than the dueling network (28.80). Therefore, the dueling network does not help on better rewards from the experiments.



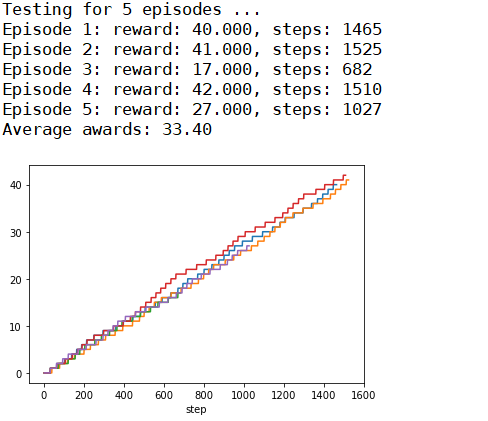
**Fig 4-12 Dueling Network Model Structure**



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



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



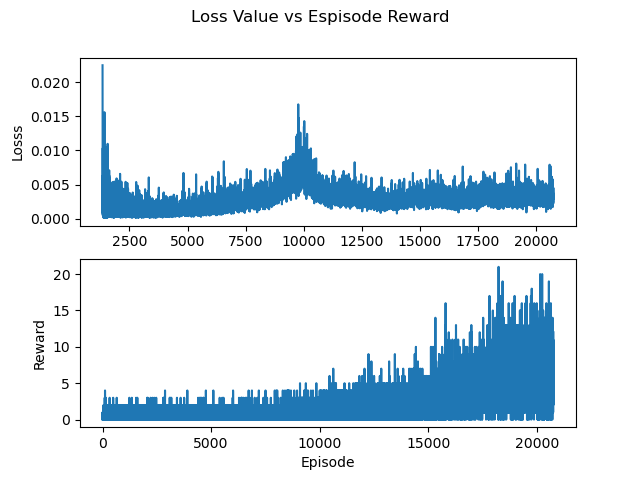
**Fig 4-15 Test Results Without Dueling Network**

4.1.5 Different Environment

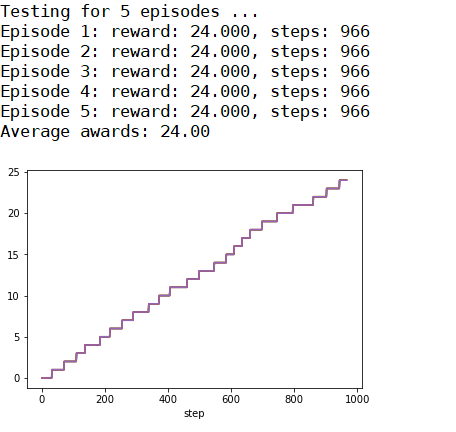
We have also tested the different enviroment “BreakoutDeterministic-v4” and compared with the current enviroment “BreakoutDeterministic-v0”.

The “BreakoutDeterministic-v0” repeat\_action\_probability is 0.25 so 25% of the time the previous action will be repeated. The “BreakoutDeterministic-v4” is 0 so it will always repeat with same action.

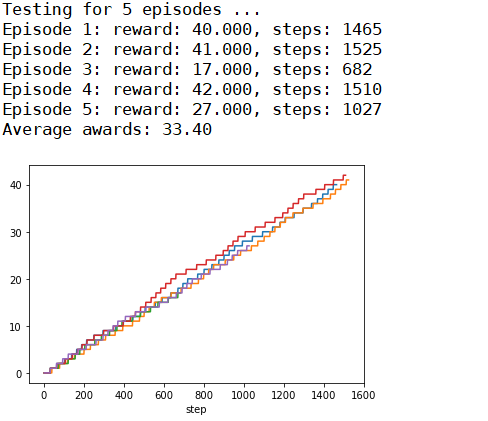
The training and test results for “BreakoutDeterministic-v4” are shown in Fig 4-16 and Fig 4-17. Compared with “BreakoutDeterministic-v0” test results in Fig 4-18, it shows that “BreakoutDeterministic-v0” has achived better rewards (33.40) than “BreakoutDeterministic-v4” (22.40) using same policy (Annealing Epsilon Greedy Policy). This is due to the uncertainty actions in “BreakoutDeterministic-v0” are 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**

The Breakout RL system is proved to be able to learn playing the game with DQN network. Even its award marks were not high, but it showed 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 the similar real-world problem, and shows the power of reinforcement learning.

5.0 CONCLUSION

**5.1 Observations & Insights**

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.

Also we do not train the RL system for longer steps such as 3,000,000 steps or 4,000,000 steps due to assignment time constrains. The larger steps can lead to better results.

Therefore, there still leaves the 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