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

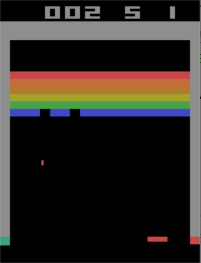
1.0 EXECUTIVE SUMMARY

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

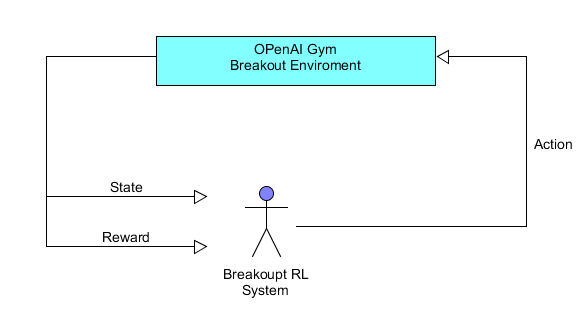
The Breakout game contains eight rows of bricks, and the player will move the paddle to hit the ball. If the ball knocks down the bricks, the player will gain the marks. If the player missed 3 times to hit the ball by moving paddle, then the player loses the turn, and new turn will begin which resets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

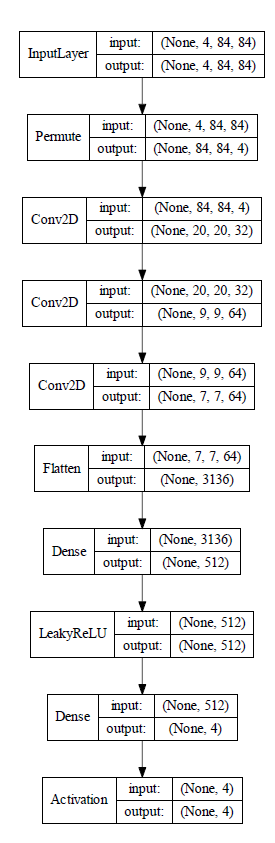


**Fig 2-2 Breakout Game Problem**

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**

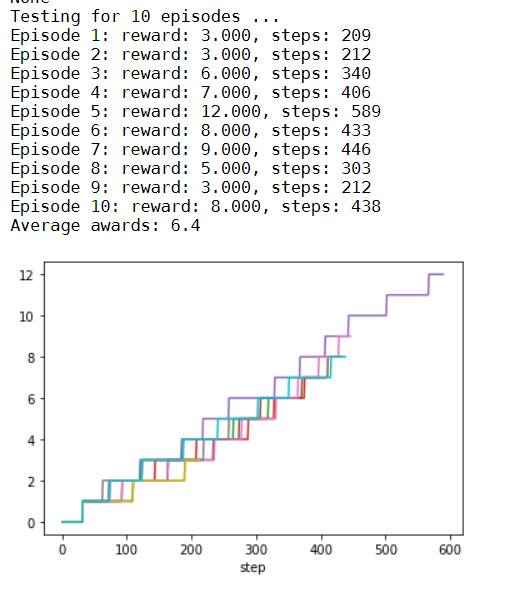
The DQN network model consists of the core of the solution design which in charge of learning the rules from the Breakout game, and also utilizing the rules to play the game.

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 was pre-built class at keras-rl package (<https://github.com/keras-rl/keras-rl>), and it connected the DQN network and Breakout Game, so it received the DQN network actions and sent them to Breakout game environment, also it transferred 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 award marks 6.4. The testing results are shown below.

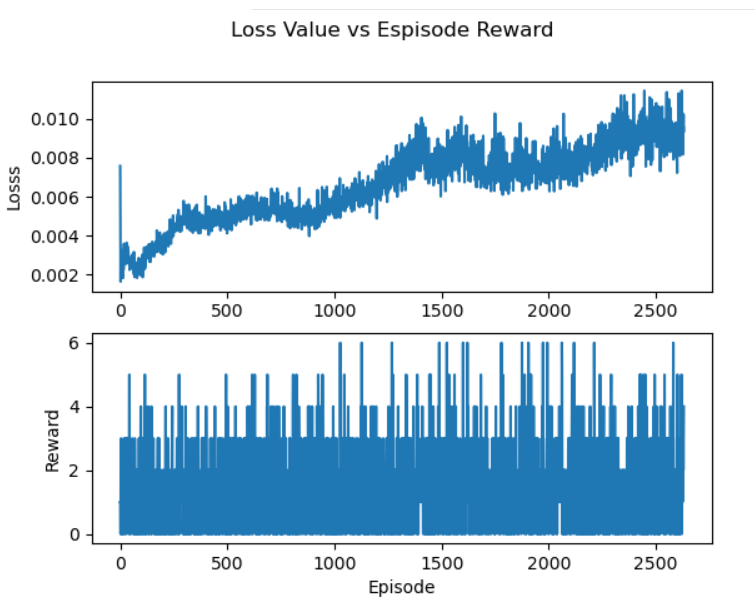


**Fig 4-1 Average Awards Result**

**4.1 Findings**

4.1.1 Kernel Initialization Parameter

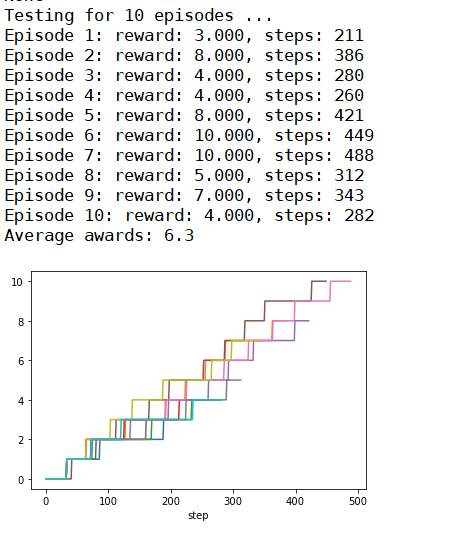
The kernel initalization was changed to “HE” normalization, but the training results did not change much. As shown below, the game awards are most below 6 during training.



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

4.1.2 DQN Agent Target Model Update

This controls how often the DQN network is updated. But after changed from 0.01 to 1000, the final testing results did not change much which are shown Fig 4-3.



**Fig 4-3 Target Model Update Parameter**

4.1.3 Policy

The policy makes the significant differents as shown in Fig 4-4. This was training results when using Epsilon Greedy Strategy with Annealing policy. And the training results when using Boltzmann Q Policy was shown in Fig 4-5.

We use eps-greedy action selection, which means that a random action is selected with probability eps. We anneal eps from 1.0 to 0.1 over the course of 1M steps. This is done so that the agent initially explores the environment (high eps) and then gradually sticks to what it knows (low eps). We also set a dedicated eps value that is used during testing. Note that we set it to 0.05 so that the agent still performs some random actions. This ensures that the agent cannot get stuck.

Linear Annealing Policy computes a current threshold value and

transfers it to an inner policy which chooses the action. The threshold

value is following a linear function decreasing over time.

policy = LinearAnnealedPolicy(EpsGreedyQPolicy(),

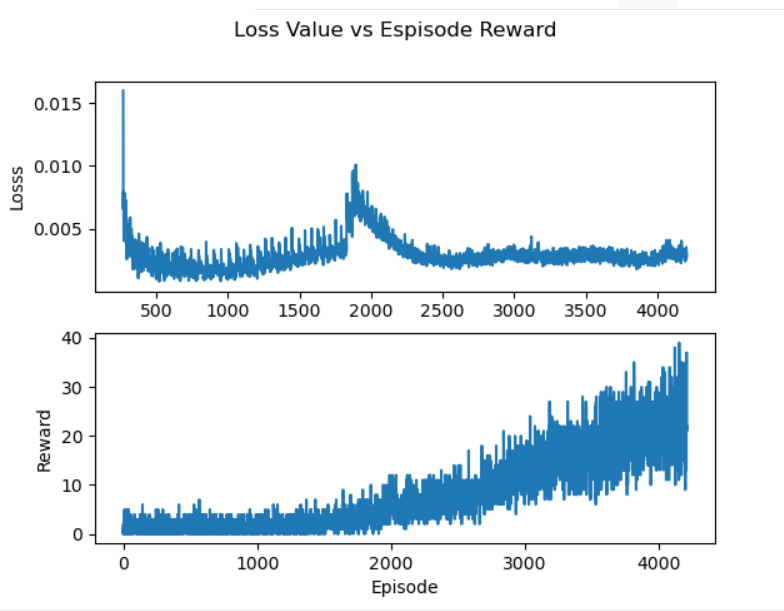
attr='eps',

value\_max=1.,

value\_min=.1,

value\_test=.05,

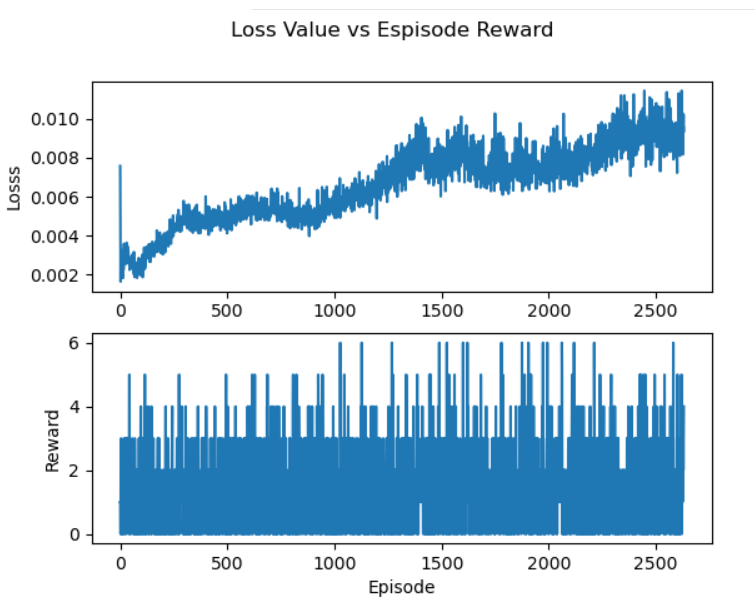
nb\_steps=1000000)



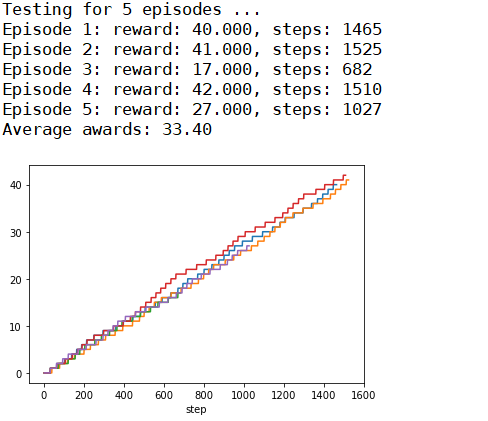
**Fig 4-4 Annealing Epsilon Greedy Policy**

Boltzmann Q Policy builds a probability law on q values and returns

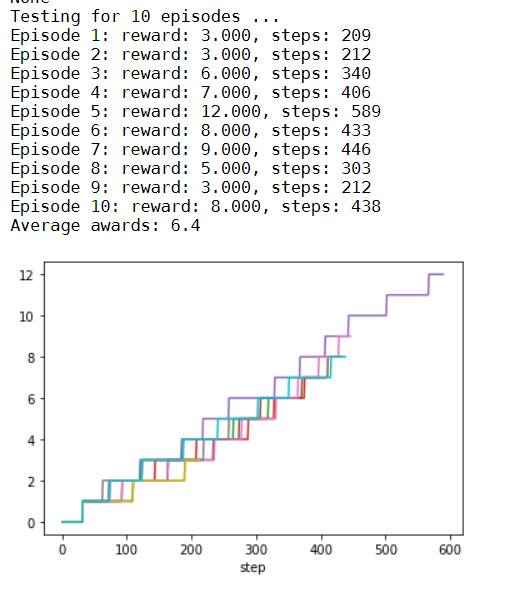
an action selected randomly according to this law



**Fig 4-5 Boltzmann Q Policy**

****

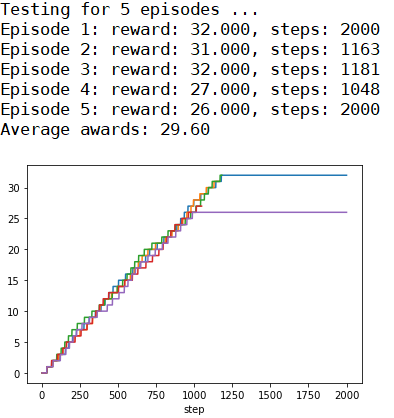
**Fig 4-6 Annealing Epsilon Greedy Policy Test Results**



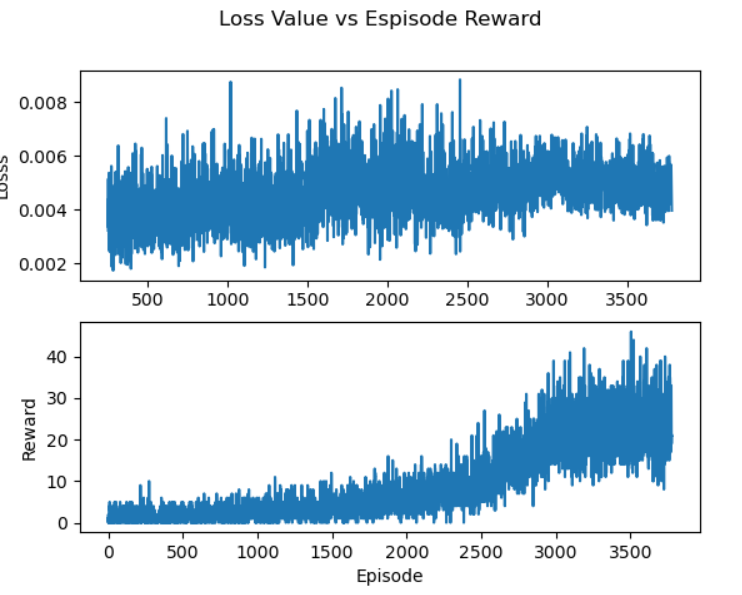
**Fig 4-7 Boltzmann Q Policy Test Results**

4.1.4 Training Steps

We have re-trained the Breakout RL system with the pre-trained weights. The re-training process take 1,700,000 steps which are the same number of steps of the pre-trained training. The test results which are shown at Fig 4-7 are imporved 2.7 % (from 28.80 to 29.60). From the training resutls at Fig 4, it shows that the episode rewards are slowly increased from the very low level (below 10) from the beginning to 2000 episodes, and reached 20 after 2500 episodes. After 3000 episodes, the rewards 30 or above are reached. Since the rewards are increased from the low to high slowly, in order to achive much higher rewards, the training steps might be increased much larger. E.g. If the first training for the Breakout RL system are 1,7000,000 steps, it may achive much better rewards if we re-train 3,400,000 steps on the Breakout RL system. Due to time constains, we have not performed the such training yet.



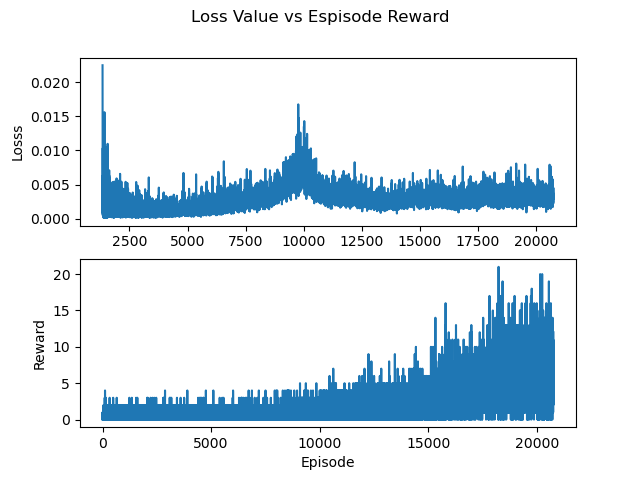
**Fig 4-8 Re-training Results**



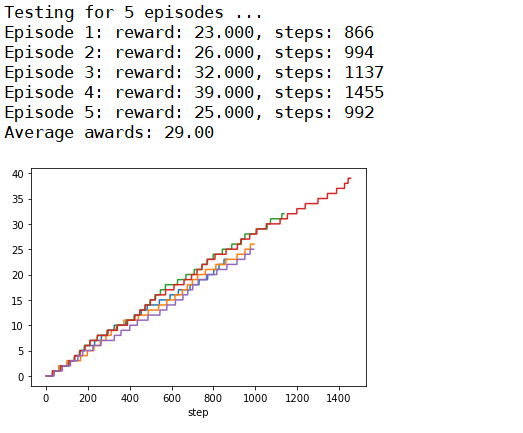
**Fig 4-9 Testing Results After Re-training**

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 4-8 and Fig 4-9. 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.



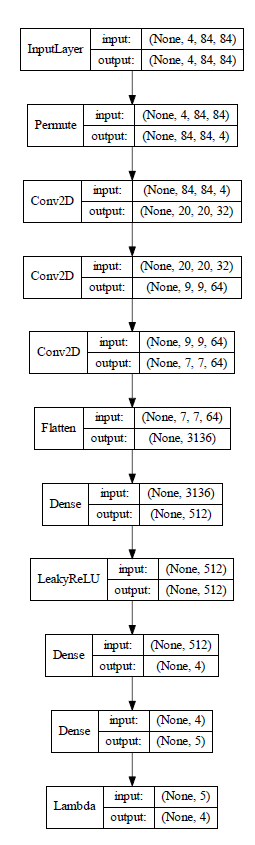
**Fig 4-10 Training Results for Game Life Lost**



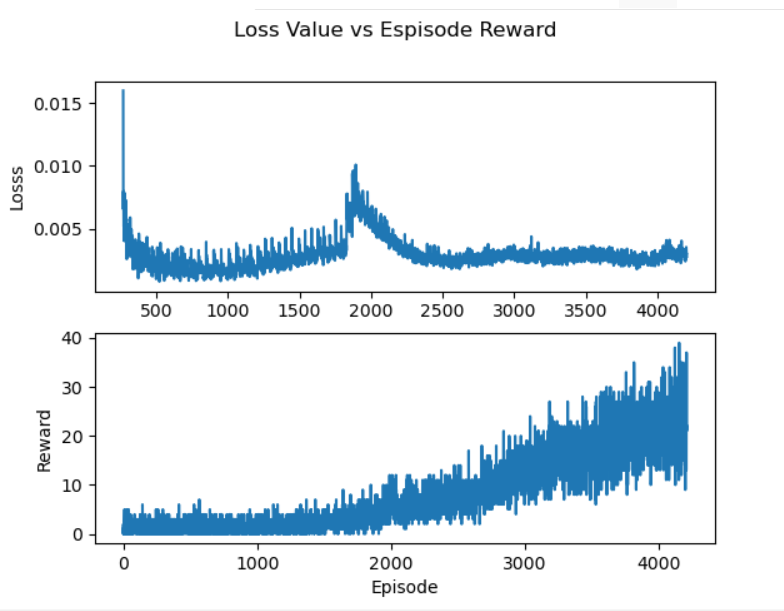
**Fig 4-11 Testing Results for Game Life Lost**

4.1.5 Dueling Network

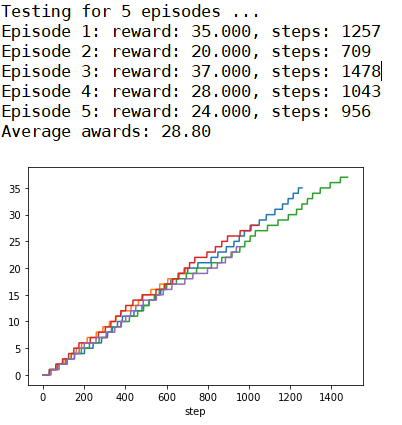
The dueling networks are pre-built option at DQN keras-rl package, and it allows the dueling network layers to be automatcally added to the existing model. We have tested the dueling network as shown in Fig 4-12. The training and testing results are shown in Fig 4-13 and Fig 4-14.



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

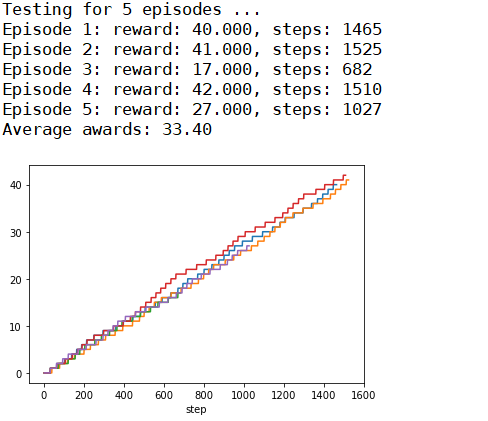


**Fig 4-13 Dueling Network Training Results**



**Fig 4-15 Dueling Network Test Results**

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



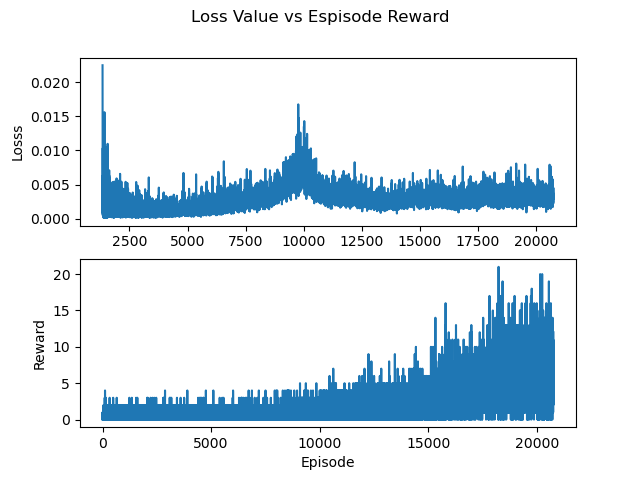
**Fig 4-16 Dueling Network Test Results**

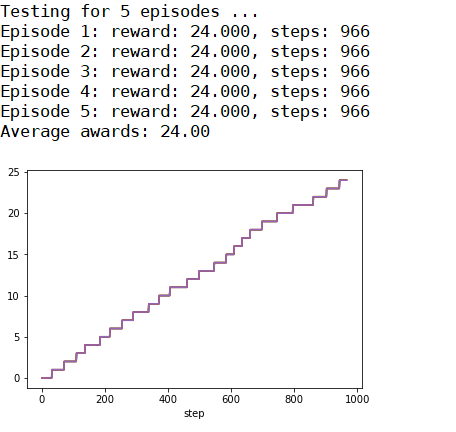
4.1.5 Different Environment

We have also tested the different Breakout enviroment “BreakoutDeterministic-v4” and compared with the enviroment settings as “BreakoutDeterministic-v4” rather than “BreakoutDeterministic-v0”.

v0 vs v4: v0 has repeat\_action\_probability of 0.25 (meaning 25% of the time the previous action will be used instead of the new action), while v4 has 0 (always follow your issued action)

Deterministic: a fixed frameskip of 4, while for the env without Deterministic, frameskip is sampled from (2,5) (code here)





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 general solution to the similar real-world problem, and shows the power of reinforcement learning.

5.0 CONCLUSION

**5.1 Observations & Insights**

To increase the trainings steps and improve with more powerful CPU; Change it to use GPU with keras-rl package; Current 1,700,000 steps training takes 12 hours;

Therefore, there still leaves the much improvement space for the Breakout RL system to enable it to achieve higher 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 gym by OpenAI: Installation instruction
* 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

1. Execution

* Training

python sls\_breakout.py train

* Show Training Results

python sls\_breakout.py plot-train

* Testing

python sls\_breakout.py test

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

File: Readme.md

* Readme file for execution setup

Folder: rl, utils

* Keras-rl packages files

Folder: history

* Training records