CS4246/CS5446 AI Planning and Decision Making

Final Project Report

Application of Reinforcement Learning in the Pacman Game

Team 53				
Name	Matric no.	Tutorial group	Email	Module code
Ng Zhi Da	A0034082R	1	e0871502@u.nus.edu	CS5446
Yap Dian Hao	A0184679H	2	e0313668@u.nus.edu	CS4246

Introduction

In the domain of AI Planning and Decision Making, Deep Reinforcement Learning (RL) is one of the most exciting and promising areas in the past few years. This is evident from the recent success of AlphaGo in 2016, which is a software powered by Deep RL engine to defeat Lee Sedol, one of the best players at Go. This is widely seen as one of the key defining moments of the advancement of Artificial Intelligence (AI). Subsequently, in 2019, Open AI Five, another Deep RL software, managed to beat the world champion in a highly complex esports game, Dota 2. This further highlights the potential of Deep RL to surpass humans in highly complex environments. This project and report aim to examine the application of Deep RL, using ***Deep Q-Learning (DQN),*** on one of the Atari games, the ***Pacman Game,*** to gain insights on the benefits of using RL to solve problems and to discuss some of the challenges and pitfalls encountered and lessons learnt.

Description of Pacman Game

The Pacman game was launched in 1982 and is an action maze chasing game, in which the player controls the main character, Pacman. Pacman's goal is to eat all the dots placed throughout the maze. However, there are four ghosts throughout the maze that are able to hunt Pacman, each with different characteristics: The red ghost chases directly after Pacman, the green and cyan ghost tries to position in front of Pacman, and the orange ghost switches between chasing Pacman and fleeing from Pacman. There are also four power-ups scattered at the corners of the maze called energizers, and Pacman can use the energizers to knock a ghost dizzy and turn blue while going in the reverse direction. Pacman can then eat the blue ghost, gaining bonus points while sending the ghost pack to the center of the maze to regenerate. However, the enhanced abilities granted by the energizers will decrease over time. Pacman can also gain bonus points through eating multiple blue ghosts. Additionally, eating a certain number of dots at different levels will drop a fruit, and can also be eaten for bonus points. Pacman has three lives, and loses a life when he is chased down by a ghost. The game ends after all the lives are used by Pacman. A screenshot of a labelled Pacman game is attached in the Appendix, Figure 7.

Project Objectives

This report will discuss the application of reinforcement learning in the Pacman game and touch on how to formulate the Pacman into a planning and decision making problem. The report will also cover how the reinforcement learning algorithm, Deep Q-Learning (also called DQN) was applied to tackle the Pacman game. A DQN model was built using Python to run and simulate the application of DQN on Pacman. The report aims to gain insights on the applicability of the algorithm through analysis on the results and outcome. Lastly, the report will touch on the challenges faced and potential further improvement.

Formulation of Pacman Game Into a Reinforcement Learning Problem

To formulate the Pacman game into an AI planning and decision making problem, the following were defined.

- Environment: OpenAI's gym environment was utilised. Specifically, the "MsPacman-V0" was used, and one of the key outputs of the game is the observation (RGB image of the screen).

- State: 4 frames of processed image, captured from the gym environment are defined as the states. (sensitivity analysis was conducted to also consider 1 frame)
- Actions: At each timestep, the agent can select one out of the 9 possible actions: 'NOOP' (do nothing), 'UP' (move up), 'RIGHT' (move right), 'LEFT' (move left), 'DOWN' (move down), 'UPRIGHT' (move upright), 'UPLEFT' (move uplift), 'DOWNRIGHT' (move downright), 'DOWNLEFT' (move downleft).
- Rewards: For each dot consumed, the Pacman will gain 10 points. Consuming an energizer will grant Pacman additional 50 points and 200 bonus points for each ghost consumed.

Application Technology Stacks

For this project, OpenAI's gym environment was used, to leverage on the game engine and visual interface that are developed by OpenAI, so the focus would be directed on the project's main objectives, to implement and examine the reinforcement learning algorithm. The implementation of the Pacman RL project is developed using Python 3. Standard libraries like Numpy (for fast matrix and arrays operations) and Matplotlib (for data visualization and analysis) were also used. Specific to building the reinforcement learning algorithm, Tensorflow and Keras were used to build the RL algorithm.

General Approach

The Pacman is a fully observable, stochastic, sequential, dynamic, single-agent, and competitive problem. While there are many RL techniques such as SARSA, Q-learning, Deep Q-learning Network (DQN) was chosen over other approaches based on a few reasons: it is a model-free approach, and the state space of the Pacman problem is huge, therefore function approximation is more appropriate. To solve the problem of catastrophic forgetting in training the model, experience replay was introduced. In addition, epsilon greedy policy was used to balance between exploration and exploitation. Instead of using only one neural network, a fixed target network was introduced to provide more stability to the learning.

For the DQN network, Convolutional Neural Network (CNN) was used to process the images, consisting of three convolutional layers with the same padding, a flattening layer, a ReLu-correction layer, and a fully connected layer. The neural network architecture can be found in the Appendix, figure 8.

Data Preprocessing

The first step is to process the image from the Pacman game. To reduce the complexity, and expedite learning, the raw observation image extracted from each time step of the game will be processed. The raw image which is based on RGB (3 dimensions) is reduced to grayscale (1 dimension). Unnecessary and redundant information such as number of lives, and score are also cropped away, as how the agent's reaction should be independent of the number lives and current score.

Building DQN

The DQN model was trained for a total of 1500 episodes (each episode consists of 3 Pacman lives). In each episode, Pacman (agent) chooses an action based on the epsilon greedy policy. If the random number chosen is less than or equal to the epsilon value ε, the agent executes a

random action (exploration). On the contrary, if the random number is greater than ε, the agent executes the action that yields the highest Q-value (exploitation).
Next, an experience replay mechanism was also included. After the agent executes the chosen action, the state, action, observed reward, and new state would be stored in the model's memory buffer. The memory buffer was implemented via a deque data structure with a fixed size, such that inserting into and popping from the queue takes constant time. For learning, a small batch from the memory buffer would be sampled randomly to fit the target neural network, and use the main neural network to predict the maximum Q-value. Subsequently, Bellman approximation was applied for the state-action values to compute the predicted Q-values and compare with the target neural network's to compute the loss. Lastly, Stochastic Gradient Descent (SGD) was applied to minimize the loss, and the weights from the main neural network to the target neural network would be sync following a certain fixed interval. The python code is uploaded to github and the link can be found in the Appendix

Hyperparameters Tuning
While implementing the DQN, there are several parameters that were explored and tuned. The learning outcome of RL is very sensitive to hyperparameters' value, and the aim of this tuning is to examine the best suited hyperparameters that will generate the maximum reward for the Pacman in the fastest time. The hyperparameters that were used in the final model are listed in Table 1. In the next section, the analysis and comparison of different values used for the critical hyperparameters will be discussed.

Hyperparameters	**Values**
State representation (frames)	4
Replay buffer's size	20000
Training batch sample size	48
Learning rate, α	0.001
Discount factor, γ	0.99

Table 1. Hyperparameters that are used for the DQN.

Outcome of Hyperparameters Selection.
Different values for the hyperparameters were explored, and the results are listed below. A moving average of 50 runs was used to visualise the results instead of the individual episode, to smoothen the fluctuation between individual episodes, so as to better identify any trends.

- State representation: In the Pacman RL application, the state is represented using frames. 1 frame and 4 consecutive frames were considered and played out. Intuitively, a state of 4 frames is more robust as it is able to capture more information such as the ghosts' direction of travel, but would incur a longer training time. After running 800

episodes for the 2 different parameter values, we observed that although the state with 1 frame's performance initially surpassed that of 4 frames, the 4 frames' state performance begins to outperform the 1 frame's state performance after around 600 episodes. Therefore, we decided to use 4 frames as the final state representation. The results are plotted in Figure 1.

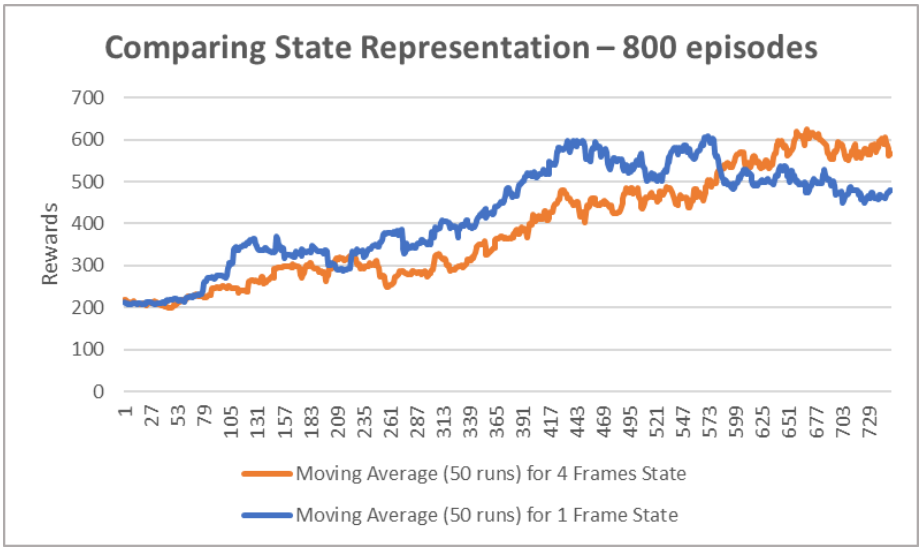

Figure 1. Graph of average rewards of 1 and 4 frames state representation

- Discount factor: A discount value of 0.99 and 0.9 are examined and results are shown below in Figure 2. The results for both scenarios are not significantly different. A discount value of 0.99 is selected eventually, as it is more commonly used in open literature for works of similar nature.

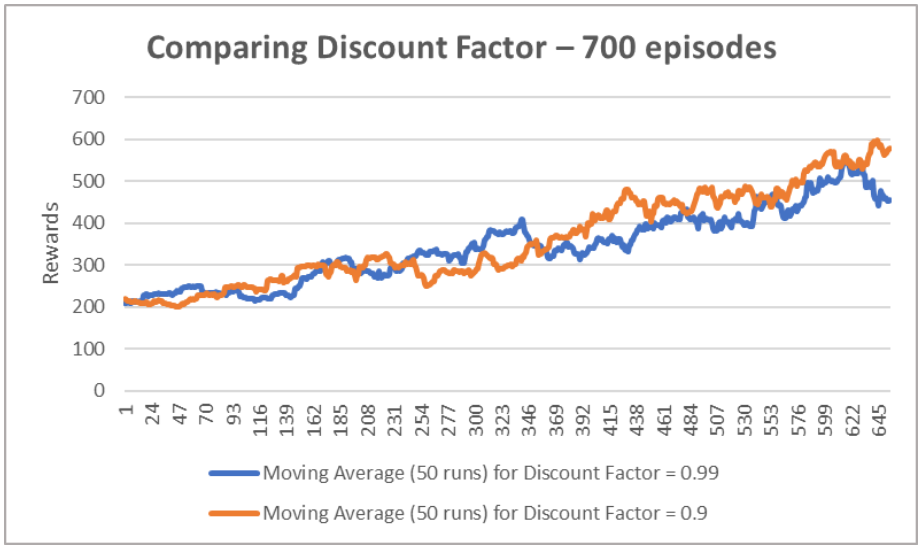

Figure 2: Graph of average rewards with discount factor 0.99 and 0.9

- Loss function: Two different ways of loss computation were considered, namely Huber's loss and the Mean Squared Error (MSE). Results showed that utilising MSE provides faster learning compared to Huber loss. The rewards are plotted in a graph in Figure 3.

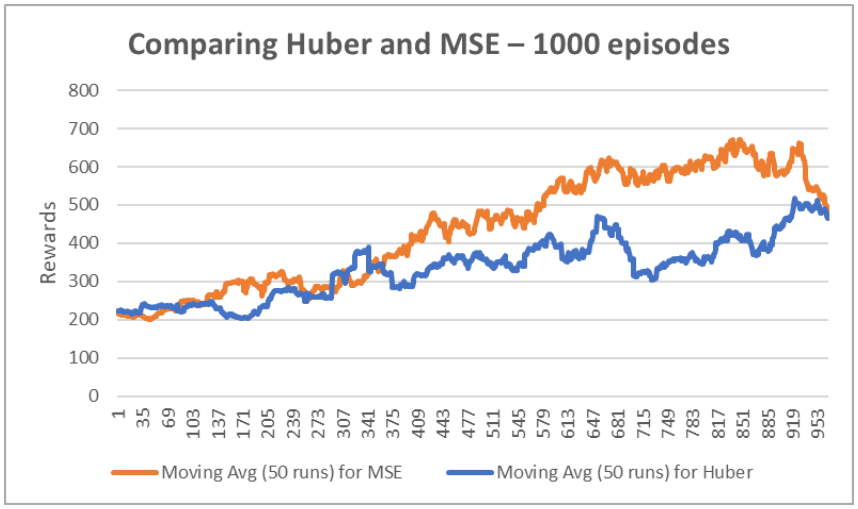

Figure 3. Graph of average rewards with Huber loss and MSE loss

- Learning rate: Two different values of learning rate were explored: 0.01 and 0.001. In RL, the learning rate is the amount of weight that is updated during training, which is also known as the step size. A good learning rate is crucial for the DQN, as a learning rate that is too small will result in longer training epochs due to the small step size, and a learning rate that is too big will cause the utility function to diverge although the training epochs required is fewer. From the graph below, we observe that the different values of learning rate outperforms each other initially, but after around 510 episodes, the learning rate of 0.001 surpasses the learning rate of 0.01. This is most likely because the learning rate of 0.01 is too large and causes the function unable to converge to local minimum. Therefore, we selected the value of 0.001 to be our final learning rate value.

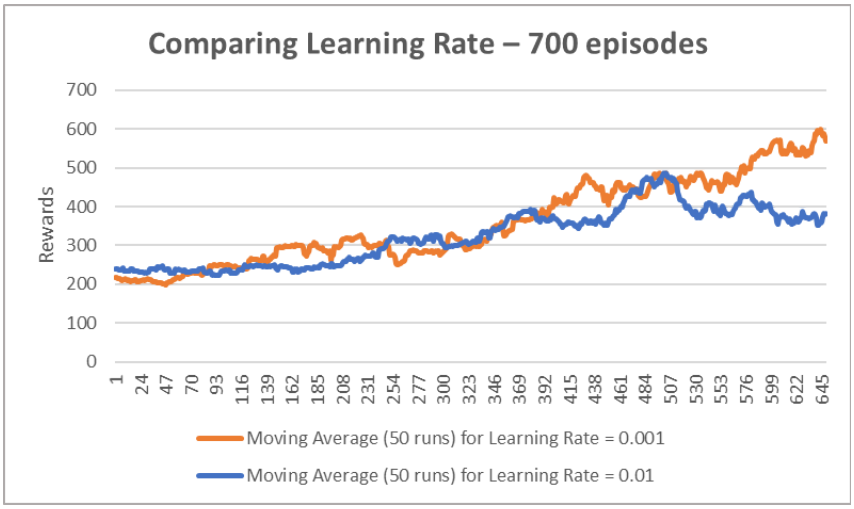

Figure 4. Graph of average rewards with 0.001 learning rate and 0.01 learning rate

Results and Analysis

After selection of the optimal parameters, the model was trained for more episodes. The obtained result, plotted as a graph is as follows. Similarly, a moving average of 50 runs was used to visualise the results. The individual episode reward can be found in Appendix. During runtime, the weights of the neural network are recorded as training progresses to allow video playback subsequently, to observe how the agent would behave at the various stages, for sensemaking.

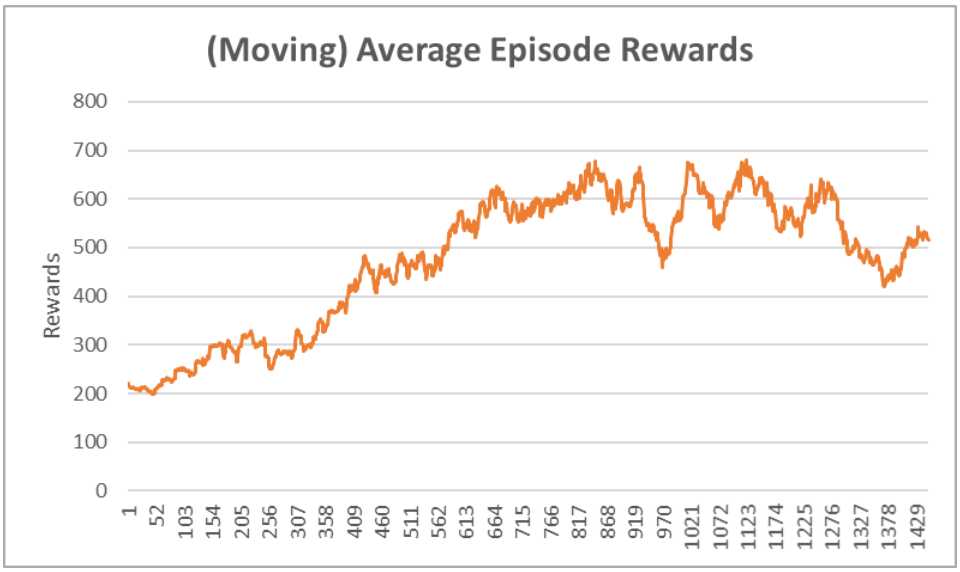

Figure 5. Graph of average rewards of 1500 runs

From the graph, it can be observed that in general, the rewards did improve over time as the total number of training episodes increased. A few key observations can be made:

- Initially, the untrained DQN starts off at an average reward of approximately 200. Through the video playback, it was observed that the Pacman exhibited random actions, and largely stayed around the starting point due to randomized behavior. This results in limited exploration of the maze, and not consuming the dots scattered around the maze.
- After 500 runs, it can be observed from the graph that the average reward is steadily increasing. The video playback also shows that Pacman started to learn to explore the maze, by moving around to eat the dots and gain the rewards.
- After 1000 runs, the reward reached a local maximum of around 700, and stabilized after which. Through examining the video playback, it was observed that by then, Pacman was found to learn a new strategy of navigating to the energizers at the corners of the map, and consuming them to gain extra points, and also the ability to defeat ghosts for bonus points. We deduced that the reward stabilized after 1000 runs and did not further increase as the epsilon decays steadily towards 1200 runs, the DQN has reached a local minimum. As the model is implemented with the epsilon greedy policy, the Pacman lacked exploration and therefore the reward did not improve further, as shown in *Figure 7* below.

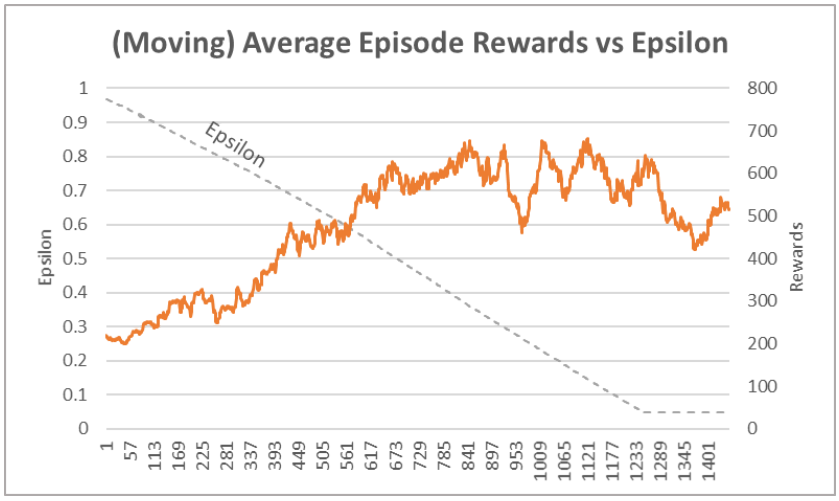

Figure 6. Graph of average rewards vs epsilon value for 1500 runs

On model's run time, it took about 40 hour of CPU time in a single terminal to run 1500 episodes of training (not inclusive of the hyperparameter tuning). This is computationally very costly, considering much time is also required to tune the parameters. Furthermore, it is apparent that there is much room for the performance of the Pacman implemented with DQN to be improved. It was observed that in many of the runs, the agent would be "stuck" in a corner (not responding to changes in the environment e.g. ghosts' positions), and also having limitations in evading ghosts and navigating the maze. Although the rewards are observed to be increasing, there is still a visible gap and against that of a human's gameplay.

Benchmarking

A few limitations were established earlier in the implementation of the DQN. To further understand the performance gap of the DQN implementation, a benchmark was done to compare against Deepmind's work in 2015. Deepmind also applied DQN on 49 Atari-2600 games (Mnih et al., 2015). Of the 49 games, Deepmind was able to achieve results at human-level or better for 29 of them. Deepmind's DQN implementation was also able to outperform the best existing RL methods, for 43 of the games, at the time of the 2015 publication. However, pertaining to the Pacman game, Deepmind was only able to achieve 13% of the human level (which achieved an average of 2311 reward per episode), and only performed slightly better than the best existing RL methods. Pacman is one of the bottom performing games that Deepmind had worked on in the paper.

In this report, the implementation yielded about 700 rewards per episode, based on training of 1.5 million frames, in contrast to Deepmind which achieved about 2300 rewards, with 50 million frames trained. The training in this project was based on a local terminal (i5 CPU Processor), which the amount of training was limited by the computation resources and time.

Potential Future Improvements

From the implementation and analysis of Deepmind results, it implies that there may be insufficiencies in the DQN approach to model the Pacman game. Unlike games like Breakout or

Pong, where the strategies involved are more straightforward, Pacman involves more complex strategies. For instance, in order to achieve high rewards, the agent needs to maximise rewards by learning to eat the dots through navigating the maze and prolonging lifespan by evading the ghosts at the same time. More complex strategies include navigating to the energizer and consuming it to grant it additional abilities at the correct timing. Pacman players would know that correct timing in eating the energizer is a key factor. One strategy would be to only consume the energizer when the ghost(s) is/are nearby, so that Pacman can be almost certain to be able to consume the nearby ghost(s) during the limited duration of enhanced ability, to maximise its rewards. Another aspect of the gameplay is the behaviour adopted after consuming the energizer. One potential strategy is to play more aggressively at the start of enhanced abilities to maximise reward, and to play safe towards the end of the enhanced ability duration, ie. to avoid being too near the ghosts when the enhanced abilities end (Tides are turned when enhanced abilities end, as now the ghost will pursue Pacman instead). All the above mentioned strategies involved longer term planning that stretched over a longer horizon within the game. Thus introducing additional components of feedback loops or aspects of memory, such as Long Short Term Memory (LSTM), which is lacking in the current model, could possibly help to achieve the agent to learn more complex strategies and improve the overall results. Alternatively, the hyperparameters can be further fine tuned, but it is assessed to have limited effects, and improvement could be marginal.

Key Challenges and Potential Pitfalls

The following are some of the challenges encountered and the potential pitfalls identified through the implementation of the Pacman DQN. These could serve as key learning points for future related work.

- Implementation results. Results of the implementation of the RL for Pacman are still not comparable to humans (even for Deepmind work), and alternate models or RL approaches should be explored and studied. It was discussed in the last section, that additional components like LSTM could be added to the current model to allow the agent to learn more complex strategies.
- Training time. It was observed that the training time required was long and computational intensive. For future work, it is suggested that policy search algorithms could be used instead of Q learning algorithms, as it may take too long to learn the Q value of all the state space. The long training time also means that it presents challenges for debugging, as abnormalities may only surface during a later phase (e.g. agent not learning). Future work can consider using more advanced debugging tools designed for machine learning.
- Hyperparameter tuning. Through performing hyperparameter tuning in the earlier section, some of the challenges and pitfalls were noted. One of the key challenges is the element of trial and error required. For example, in determining a suitable epsilon value, it was only after running, one could have a sensing of how the agent interacted with the chosen epsilon figure / features. Coupled with the long runtime, sufficient time would be needed to be catered upfront for hyperparameter tuning. Earlier, in the conduct of hyperparameter tuning, the hyperparameters' values were based on the results from the first 1000 runs. Ideally, the hyperparameters tuning should be done based on as

many runs as possible. However due to limitations on computational resources, there is a constraint on the number of runs conducted.

- Challenges in OpenAI gym environment. Firstly, there is a lack of documentation of libraries in OpenAI's Pacman gym environment. Secondly, due to bugs in the underlying environment, the ghosts appear to be flickering at times, and this could potentially degrade and affect the learning, due to incorrect representation of the states.

Conclusion and key learning points.

In conclusion, the report covered the formulation of the Pacman game into a Reinforcement Learning DQN problem. The project also covered the building and implementation of the Pacman DQN model using Python, which successfully trained the Pacman agent to execute behaviours or learn strategies to improve rewards in the game. Lastly, the report also covered some of the challenges and pitfalls, and potential improvements for any future work.

Through the above implementation and analysis, it is not hard to understand that, while RL is a very promising domain, there are still many limitations. Some are evident in the project, such as long training time. From the Deepmind example, one can also note that there is no one-size-fit-all solution. The same approach used on 2 Atari games (e.g. Breakout vs Pacman) can yield very different outcomes. As such, the trained model on one problem/game cannot be easily transferred, and coupled with the long training time, it presents many challenges in applications for real world use cases.

In real world problems, the problems are more likely to be more complex in several magnitudes, compared to the Pacman game, and often the environment is usually partially observable. The challenges discussed would be even more pronounced. Nevertheless, the project members look forward to many more breakthroughs, as RL continues to be one of the top areas that are currently researched on.

Effort and Initiative

The members of the team worked closely together and contributed evenly throughout the project. Regular meetings were held to discuss the tasks ahead, challenges faced and brainstormed on how to address the problems faced. While for each of the tasks, the work was split into 2 as much as possible, the team had clear leads for the various tasks to identify who is responsible for the particular task outcome. Zhi Da led the problem formulation, model development, benchmarking and project presentation. Dian Hao led the research of the techniques, analysis of results and the final report.

References

Xu, Y. (2019). Automating Pac-man with Deep Q-Learning: An Implementation in Tensorflow Automating Pac-man with Deep Q-learning: An Implementation in Tensorflow. | by Adrian Yijie Xu | Towards Data Science

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Rusu, A. (2015). Human Level Control through Deep Reinforcement Learning

Appendix

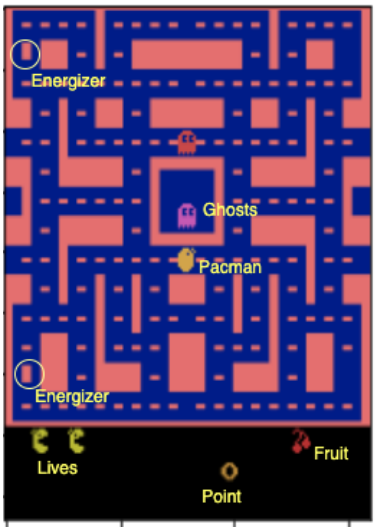

Figure 7. Sample screenshot of Pacman game

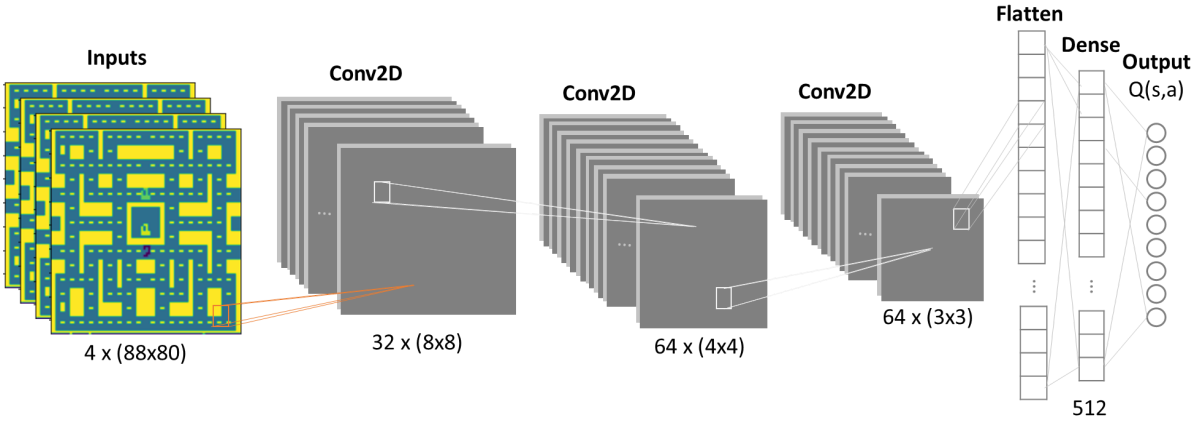

Figure 8. Diagram of layers used in our DQN neural network

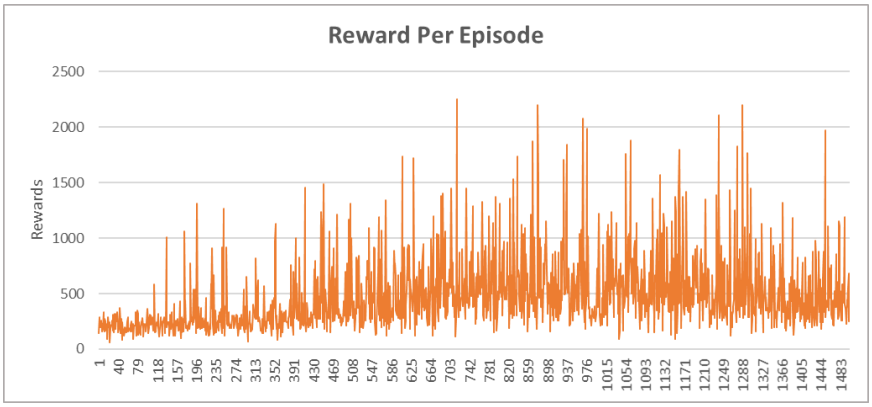

Figure 9. Reward Per Episode

Project Code link: CS4246-CS5446-Team-53/Project (github.com)