Learning to play Super Mario Bros. using DQN, DDQN and Dueling DDQN

Zachriah Jose
Department of Computer Science
University of Bath
zj401@bath.ac.uk

Konstantinos Azas
Department of Computer Science
University of Bath
ca615@bath.ac.uk

George Aristodemou
Department of Computer Science
University of Bath
ga455@bath.ac.uk

Anastasios Karageorgis
Department of Computer Science
University of Bath
ak3289@bath.ac.uk

1 Problem Definition

The aim of this project is to train an agent capable of completing the first stage of world 1 of the Super Mario Bros. game. The environment in which all agents implemented were trained is based on the gym environment, as it is presented in (GitHub citation). The state used is a greyscale image and has dimensions of 4x84x84. After each state, the agent selects an action based on its training policy, which is then given as input to the environment, that then generates the next state. There where many choices for the action space to be used. It was determined that the Right Only action space, consisting of 5 possible moves (NOOP, Right, Right + A, Right + B and Right + A + B) would be used for all implementations. The reward function of the environment considers v which is velocity, c the time penalty and d the death penalty. The reward function satisfies the equation $r = v + c + d$. A more detailed description of the problem domain can be found in appendix 8.1.

2 Background

2.1 Q-learning background

Using the basic structure of the Q-learning algorithm would be a kick start to implement for this problem. Q-learning is an off-policy method which means that the agent utilizes an ϵ-soft policy to randomly explore the environment or exploit what it already knows to try and maximise its rewards. While doing so, the agent collects experience in a form of (state, action, reward, next state) and is learning the optimal strategy. After some time, the agent now is able to understand the environment based on the estimated q-values. In Q-learning these values are updated by taking the maximum q-value of the next state for all possible actions the agent can choose [6]. The update is formulated as:

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big(R + \gamma \max_{a} Q(S^{'}, a) - Q(S, A)\big) \tag{1}$$

2.2 Neural Networks

Due to the complexity of the Mario environment, tabular methods like Q-learning has some limitations regarding memory and capability. First of all, it is impossible to store the huge amount of state-action pairs that have been explored and it forces the agent to visit all state-actions so that a value estimate is induced for each one. This needs to be expanded in more robust methods, using function approximations. Function approximation is a technique used for large and complex enough environments

that basically induces a parameter $\boldsymbol{\theta}$ used to store information from a part of the state space. So, the state space can be thought of as being split into individual experience partitions and as the agent starts from the initial partition, the trajectory is being stored in the form of (S_t, A_t, R_t, S_{t+1}). This helps in reducing memory consumption, since only the model parameters need to be stored and also there is no need for the agent to visit all state-actions. This is due to the fact that the function can be used to better generalise learning to unseen state-actions [2]. The simplest approach for function approximation is by using linear function approximation. In this case, each pixel from the image has its corresponding weight that must be learnt and taking a linear combination of those, an output is computed. There are some limitations though when a 2-D image needs to be converted to a 1-D vector. Linear function approximation loses the spatial information of an image [1]. An additional disadvantage of this approach is that it is far too simplistic. For example, for our considered problem a linear function would be used for each action. The total number of parameters in this case would be state*actions (4x84x84x5 for right only movement or equal to 141120). For comparison the DDQN approach implemented uses upwards of 1.6 million trainable parameters. Although the number of parameters is not an absolute measurement for complexity it is a good indicator showcasing the disparity between the two methods. To resolve both of these issues, a convolutional neural network will be used instead.

2.3 Algorithms

For solving this problems, three separate methods have been implemented. These are Deep Q-Networks (DQN), Double Deep Q-Networks (DDQN) and Duelling DDQN.

2.3.1 DQN

A deep Q-network is a multi-layered neural network that for a given state, computes the corresponding Q-values for each action. The main difference from Q-learning is the induction of a neural network instead of a Q-table which is suitable for large environments like ours, due to the implementation of function approximation. Furthermore, it is also suitable for the reasons discussed in section 2.2. Some limitations though, it uses the same values both to select and to evaluate an action [8]. It is not always the case that the best action for the next state is the one with the highest Q-value. In the beginning of training, when the information is limited, wrong actions end up having the highest Q-value. This leads for value estimates to be overestimated. This algorithm is performed in [3] and also observed that DQN is not performing that well. This has also been observed from our results as well, which are illustrated in section 4.

2.3.2 DDQN

This algorithm tries to prevent the limitations occurring in DQN. It achieves this by decoupling the action selection from the q-value estimation [1]. Action is selected from the online network and the corresponding q-value is retrieved from the target network. Due to this change, the overestimation of the q-values is significantly decreased. Thus, leading to an advantage over DQN. This is the only change between the two algorithms. Additional evidence is depicted in [3] where the learning curve of DDQN is significantly better than DQN. Some limitations though, for most of the states it might be unnecessary to estimate the q-values for all possible actions [3]. This can lead for the agent not to be sure to distinguish between a 'good' and a 'bad' next state [9]. Hence, value estimates are again overestimated.

2.3.3 Duelling DDQN

This algorithm tries to prevent the limitations occurring in DDQN. It achieves this by inducing a value function and an advantage function separately. With the aid of the advantage function this will help the agent to avoid what is considered a 'bad' next state. Also, it is possible to learn which states are valuable without having to learn the effect of each action at each state. This is useful for states where the action taken does not affect the environment in a relevant way [7]. In the context of Super Mario Bros., evaluating different actions while in a wide-open area free of enemies is not necessary, whereas it is important when an enemy is approaching, or a pit is nearby [3]. Further evidence is again provided in [3] where it seems that this algorithm outperforms the rest. There is also evidence from the learning curves in Section 4.

3 Method

3.1 Deep Q-Network

The DQN is a deep reinforcement learning algorithm which uses neural networks as a function approximator between the state of the environment and the Q-values for each action that could be taken. The DQN algorithm, that was used for the project uses two neural networks, an "online" network for predicting Q-values for each state-action pair and a "target" network which gives the Q-values that acts as a target for training the online neural network. The target function is defined as:

$$Q_{target}^{DQN} = R_{t+1} + \gamma \max_a Q_{target}(S_{t+1}, a) \tag{2}$$

where,

- γ = Discount factor
- R_{t+1} = Reward obtained for taking the action a_t at state S_t
- S_{t+1} = The next state
- $\max_a Q_{target}(S_{t+1}, a)$ = the maximum Q-value for a state-action pair obtained when the next state is passed through the target network.

For the backpropagation, the loss function is shown below, while the updating scheme can be seen in image 1.

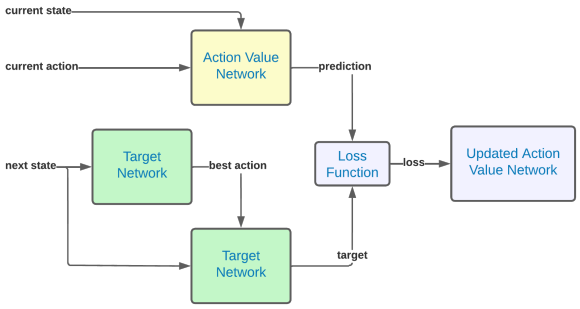

Figure 1: Updating scheme of DQN

3.1.1 Loss Function

The loss function mainly used for Deep Reinforcement learning is the Huber loss instead of MSE. The reason is because with MSE, large errors are encountered which implies large updates in the weights [4]. Using the Huber loss, large errors can be penalized and it is formulated as:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, \text{if } |y - \hat{y}| \leq \delta \\ \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2, \text{otherwise} \end{cases} \tag{3}$$

where y is the target value seen in 2, and $\hat{y}$ is the predicted value ($Q_{online}(S_t, a)$). The action a is chosen as:

$$a = \text{argmax}_a Q_{target}(S_{t+1}, a) \tag{4}$$

An important point to note is that if the online network and target network is updated at the same time while training, it would lead to oscillations in the result. Therefore, it means that at every step of training, the Q-values shift but also the target values also shifts. So, the prediction is getting closer to the target but the target is also moving. This is fixed using the fixed Q-targets method proposed by Google Deepmind [5]. The parameters of the online network which is used for predicting the best action is updated every small interval of steps while the parameters of the target network are frozen. After a certain number of steps, parameters of the online network are copied in order to update the target network.

3.2 Double Deep Q-Network (DDQN)

As mentioned in the Background section 2.3, the main disadvantage of using DQN is the overestimation of the Q-values. DDQN is similar to DQN in the sense that it uses a neural network as a function approximator for the Q-values and uses values from two different neural networks for calculation of loss while training. The only difference between DQN and DDQN is in the way the target value is calculated. The target function is defined as:

$$Q_{target}^{DDQN} = R_{t+1} + \gamma Q_{target}(S_{t+1}, \text{argmax}_a Q_{online}(S_{t+1}, a)) \tag{5}$$

The updating scheme of DDQN is illustrated below in image 2. The next state is passed through the

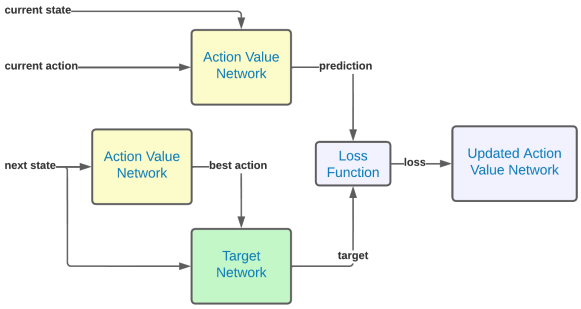

Figure 2: Updating scheme of DDQN

online network and the action corresponding to the highest Q-value is selected. Then the next state is passed through the target network and the Q value corresponding to the action estimated before is found. This $Q_{target}(S_{t+1}, \text{argmax}_a Q_{online}(S_{t+1}, a))$ is used for calculating the target value in DDQN. Backpropagation and $Q_{online}(S_t, a_t)$ are calculated similarly as in DQN.

3.3 Duelling Double Deep Q-Network (Duelling DDQN)

The Q-values estimates how good it is to be at a particular state and taking an action at that state. In duelling DDQN, the Q-values are decomposed into $V(s)$, the value of being in that state and $A(s, a)$, the advantage of taking that action at that state. The Q-value is formulated as:

$$Q(s, a) = V(s) + A(s, a) \tag{6}$$

Therefore the neural networks have two branches, one for calculating the value and one for calculating the advantage. Concerning the aggregation layer of the neural network, we would like to combine the value and the advantage to generate Q-values for each action at that state. But, just adding both of them would result in an issue of identifiability, that is given $Q(s, a)$, we won't be able to find $V(s)$ and $A(s, a)$. This becomes a problem during backpropagation. To avoid this problem, we can force our advantage function estimator to have zero advantage at the chosen action. To allow backpropagation we can re-formulate the Q-value as:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{n} \sum_a A(s, a) \tag{7}$$

where the last term indicates an average of the advantage over all the actions.

Therefore, this architecture helps in accelerate the training. And it helps in finding much more reliable Q-values for each action by decoupling the estimation between two streams. The target function and the loss function of Duelling DDQN is exactly the same as DDQN. Since the neural network outputs Q-values just like the DDQN neural network, the methodology used for DDQN is applicable for forward propagation and backpropagation.

4 Results

In this section of the report, the results of the implemented agents are presented. The agents presented are a random agent as a baseline, a DQN agent, a DDQN agent and our proposed Dueling DDQN

agent. All agents were trained for 1000 episodes. However, after training DQN and DDQN, it was decided that a change in hyperparameter values would improve performance and thus, Dueling DDQN was trained with a different hyperparameter set. A complete summary of the experimental details can be found in appendix 8.2. The undiscounted returns for all agents can be seen in graph 3. It is clearly evident that this graph cannot be easily interpreted. This is due to the fact that the environment has many local minima with the selected action space. When Mario is randomly exploring, he can die, get stuck or run out of time resulting in much lower rewards. In order to better visualise the results, a moving average over 100 steps was utilised. The moving average results for all agents can be seen in graph 4. As can be seen from this graph, the random agent shows the worst performance. Since no learning is happening, performance will remain the same throughout. The DQN agent achieved the second to worst performance. This goes to show that, due to the disadvantages of DQN, this approach is not well suited for this problem. The DDQN agent shows some improvement over DQN, which is expected as the decision making, and evaluation of the Q-values are more decoupled. Finally, the best performance by far was achieved by the Dueling DDQN agent. The learning curve for this agent is steeper when compared to DDQN, indicating more efficient learning. The graph also shows higher maximum values, meaning that the learning of the Dueling DDQN agent is also better. Finally, graph 5 shows the completion rate for the Duelling DDQN agent. It is clearly evident from this graph, that Dueling DDQN is very effective at completing the task.

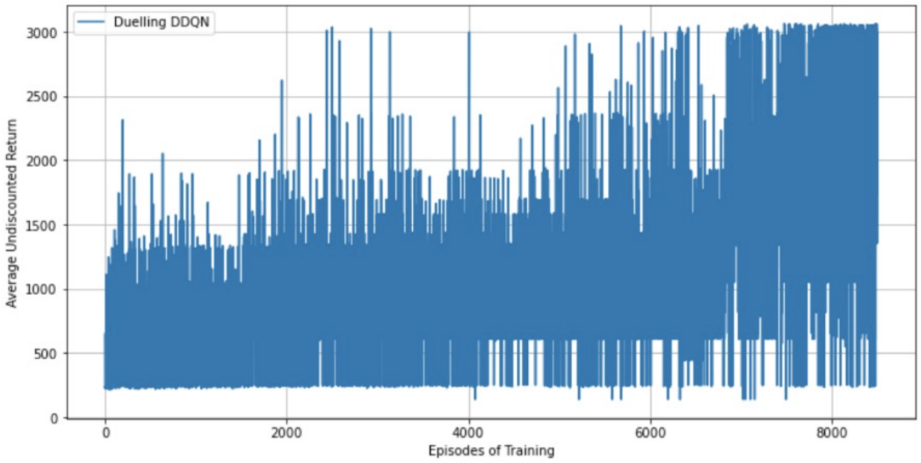

Figure 3: Learning curve of DDQN

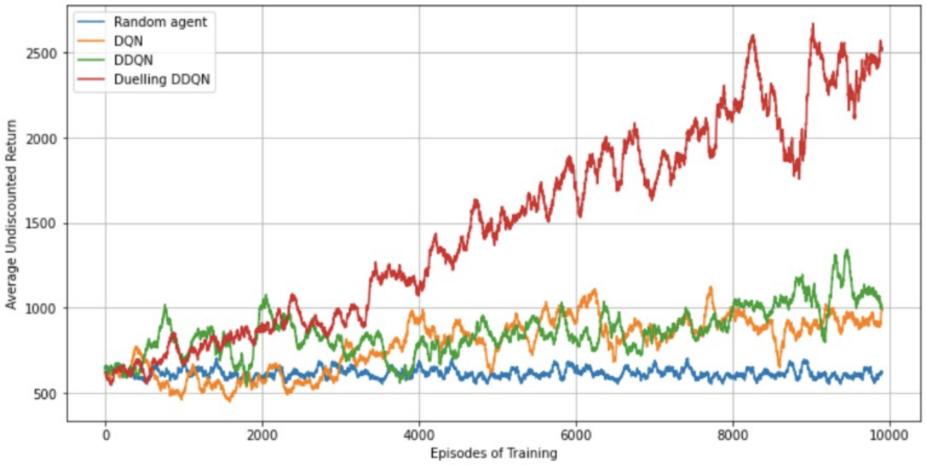

Figure 4: Moving averages of algorithms

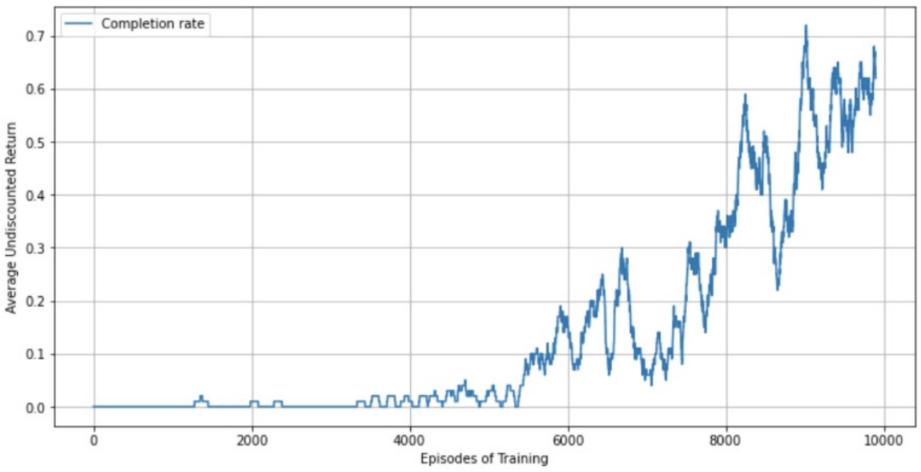

Figure 5: Moving Average of rewards for all agents

5 Discussion

In this project Reinforcement learning was applied to create an agent capable of reaching the end of the first level of World 1 for the game of Super Mario Bros. After having read several relevant works and learning about the strengths and weaknesses of various methods, DQN, DDQN and dueling DDQN agents were implemented to solve the problem. The results indicate that dueling DDQN is the best approach for this problem followed by DDQN and DQN. As evidenced by the graphs presented in the Results section, dueling DDQN has the steepest learning curve, which indicates more efficient learning when compared to the other two methods. It is also clearly evident by the DQN learning curve, that this model is too simple and unable to solve the problem. These results are in agreement with the relevant literature presented in the Background section of this report. The DDQN method, reduces the overestimation of the Q-values, which is the main drawback of DQN, and dueling DDQN reduces the overestimation more, due to the extended CNN used (which calculates both the value of the state in question and the advantage of selecting each action at that state), thus yielding further increases in performance.

6 Future Work

Keeping in mind the main disadvantage of DQN expressed in paper (citation) which is that the Q-values are overestimated, a DDQN approach was adopted, which was then expanded to Dueling DDQN. Even though it is true that DDQN is an improvement, it does not completely solve the underlying problem that leads to this overestimation. The two networks used are still coupled due to syncing. In order to eliminate this problem a new algorithm to be called Two Agent DDQN is proposed as future work. Just like in tabular Double Q-Learning, two agents will be used. The experience will be evenly split among them. Instead of choosing an action based on the action-value network of the agent and getting the corresponding Q-value from the target network of the same agent, the action-value network of the second agent will be used. In doing so, the decoupling of the two steps is guaranteed. This implementation, however, lies outside the scope of this project, and is thus included here as a possible future implementation to further improve performance. Additionally, this method could be combined with Dueling.

7 Personal Experience

Upon the completion of the project various experiences and skills were gained, along with a few difficulties. The group became proficient in understanding how different variations of the Q-Learning algorithm, such as DQN, DDQN and Dueling DDQN work. Moreover, this coursewrok allowed the team to discover the best practices for compiling a report on a project of this scale. In addition, one of the biggest experiences gained was the group using reinforcement learning to solve a real complex problem for the first time, giving us the necessary foundations to enter the workplace after university. It was also a surprise to the group on how much effort and research is needed concerning reinforcement learning in the gaming industry and the speed-running community, not just for Super Mario but for all sorts of games. Furthermore, one of the biggest difficulties faced was trying to understand relevant codes concerning the projects as most of them contained multiple classes and also the use of libraries such as Pytorch with which nobody in the group had experience with.

References

[1] aravindpai. Ann vs cnn vs rnn - analyzing 3 types of neural networks in deep learning. 2020, https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/.

[2] George Berlak. What are some reasons why we might choose to use function approximation in rl?, 2020, https://www.quora.com/What-are-some-reasons-why-we-might-choose-to-use-function-approximation-in-RL.

[3] Sam Donald. Playing super mario bros. with various deep reinforcement learning architecture.

[4] Joshua Evans. Dqn deep-dive. 2021, https://moodle.bath.ac.uk/pluginfile.php/1100399/course/section/174649/Lecture%2014%20-%20DQN%20Deep%20Dive.pdf.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] Mohit Sewak. *Deep reinforcement learning*. Springer, 2019.

[7] Thomas Simonini. Improvements in deep q learning: Dueling double dqn, prioritized experience replay, and fixed. . . . 2018, https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/.

[8] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[9] Chris Yoon. Dueling deep q networks, 2019, https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751.

[10] Feng Yuansong. Madmario. 2021, https://github.com/yfeng997/MadMario.

8 Appendices

8.1 Appendix 1

In order to improve the effectiveness of the agents, the game images are prepossessed. The image from the gym environment (original image before preprocessing can be seen in image 6) is converted into grayscale and then its size is reduced to 84x84. This is done to reduce the size of the state space. In addition to this change, the frames are also stacked, so that every state is a set of four consecutive frames. Assuming every frame corresponds to a time step, then a state as is defined above is created every four time steps. The difference in time steps between two consecutive states, thus becomes four. The preprocessing can be seen in image 7. However, the agent still needs to perform an action at each frame. Therefore, the same action is selected for all four frames in each state.
This problem is a very complex one, not only because of the large action space, but also due to the plethora of actions available to the agent. As such, there are various actions spaces. These include, Right Only, Simple and Complex Movement action sets. Action spaces affect the complexity of the problem, and for this project the Right only action space is used. This action space includes five different moves: NOOP (meaning no action is performed), Right (move to the right), Right + A (move to the right and jump), Right + B (move to the right and fire) and finally Right + A + B (move to the right, jump and fire). Although the B button makes Mario throw a fire ball, if Mario is not in fire Mario form (by consuming a fire flower and avoiding hits), holding down the B button will instead make him move faster.
The reward function for the gym environment is composed of three variables: the change in the horizontal position between this state and the next v, the difference of the in-game clock between the two states c and finally a death penalty d. Death is caused by touching an enemy, falling into a pit, running out the in-game clock or causing the environment to kill Mario because he got stuck. The enemies for the chosen stage, are Goombas 8, which are little mushrooms walking from the right side of the screen to the left and Koopa Troopers 9 that are turtles also moving in the same fashion. Both types of enemies will change direction if they come in contact with an obstacle such as a pipe. If a Koopa Trooper is hit from above by Mario, it turns into a Koopa shell and if hit again, depending on the position of Mario, it will move towards either the left or the right with great speed. Its direction can be changed by coming in contact with obstacles and if it hits other enemies, it will kill them. Finally, if this shell then hits Mario, he will die. The reward r satisfies the equation $r = v + c + d$ but is clipped to be in the range $(-15, 15)$. Since we stack the frames, the reward for each state is the sum of the rewards of each of the stacked frames, thus the total reward lies in the range $(-60, 60)$.

All images used in this appendix are from the paper (cite Marlo).

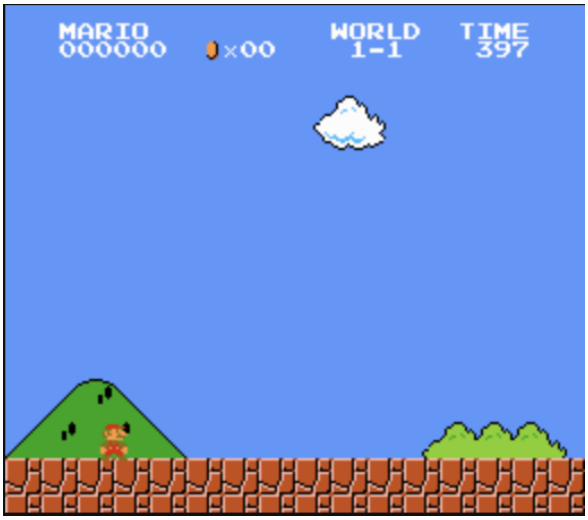

Figure 6: Original environment image before preprocessing.

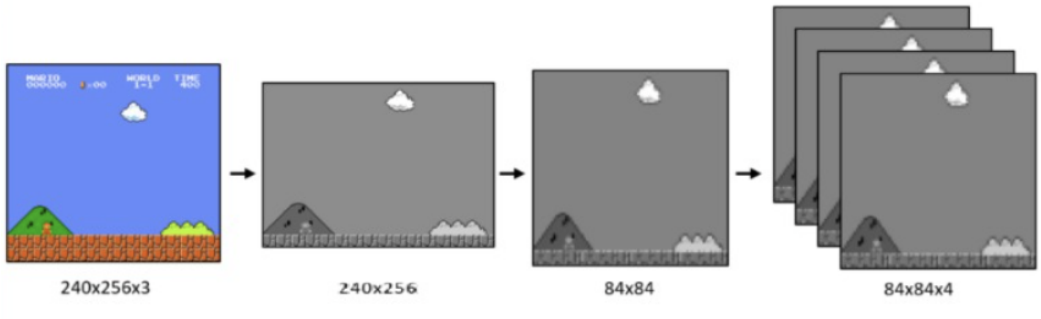

Figure 7: Preprocessing steps.

Figure 8: Goomba.

Figure 9: Koopa trooper.

8.2 Appendix 2

8.2.1 Pre-Processing

In appendix 8.1, the pre-processing is explained analytically. The original image is converted to gray scale and this is replicated 4 times as a stack of 4 channels. The dimension of each output is 84x84x4

8.2.2 Model Architecture

Both the DQN and DDQN algorithms used the same network for the online and target networks since the implementation of the algorithms are similar and the only difference is in during the calculation of loss at each point of training. The architecture used comprises of the following layers, all of which use a (Rectified Linear Unit) ReLU activation function;

- Input: 84x84x4 pre-processed frames
- Hidden layer 1: 2D Convolutional Layer with output size 32 and kernel size:8 with stride 4.
- Hidden layer 2: 2D Convolutional Layer with output size 64 and kernel size 4 with stride 2.
- Hidden layer 3: 2D Convolutional Layer with output size 64 and kernel size 3 with stride 1.
- Hidden layer 4: Fully connected linear layer with input size 3136 and output size 512.
- Output layer: Fully connected linear layer, with input size 512 and output size based on action space size.

The architecture was based on the MadMario PyTorch Reinforcement tutorial [3].

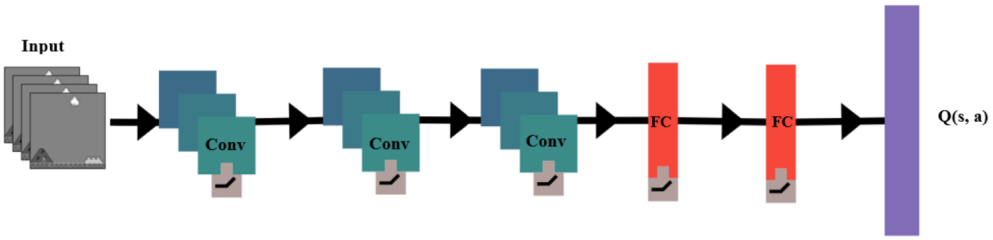

Figure 10: Network for DQN and DDQN

8.2.3 Model Architecture for Duelling DDQN

The initial layers of the neural network is same as the DQN and DDQN architecture. The fully connected layers are replaced with two branches, namely the value branch and the advantage branch. These two branches are then combined together to output a set of Q-values for a certain state [3]. The architecture is shown below:

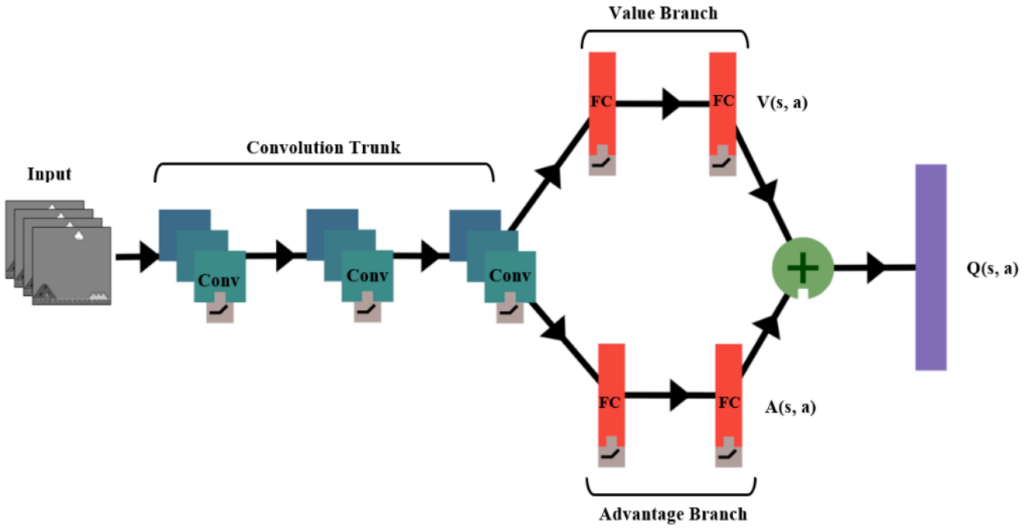

Figure 11: Network for Duelling DDQN

Value Branch:

- Fully Connected Layer 1 : Linear layer with input size 3136 and output size 512.
- Fully Connected Layer 2 : Linear layer with input size 512 and output size 1

Advantage Branch:

- Fully Connected Layer 1 : Linear layer with input size 3136 and output size 512.
- Fully Connected Layer 2 : Linear layer with input size 512 and output size based on the action space.

8.2.4 Project Implementation

Initially, the agent is allowed to accumulate experiences with a higher exploration rate with the exploration rate starting at 1. It is reduced at a rate of 0.993 per episode with a minimum exploration rate of 0.01. The experiences are stored in a replay buffer of size $30,000$ which is used for sampling experiences at a batch size of 64 for training the neural network. For the first $30,000$ steps the agent takes random steps and accumulate experiences. After that it is trained for a further 10000 episodes. At the start of each episode, the environment is reset and agent takes an action based on the highest Q-value out of those given by the online neural network for that particular state. The information sent by the environment after taking that action is stored in the replay buffer where newer experiences replace the older ones. Every 3 steps, the parameters of the online neural network are

updated using the loss function between the online Q-values and the target Q-values. And, every 5000 steps, the parameters of the online network are copied in order to update the parameters of the target network.

8.3 Appendix 3

8.3.1 Hyperparameters

The architecture and the hyperparameters for DQN and DDQN were inspired from the Mad Mario project by [10].The architecture for the Duelling DDQN and the size of the replay buffer were influenced by the MaRLo project [3]. The other hyperparameters were influenced from the results of DQN and DDQN and were devised by the group to allow more exploration by the agent such that the exploration rate becomes a minimum of 0.01 at around 5000 episodes. The learning rate and the sync rate were same as the DQN and DDQN algorithms which were obtained from the MadMario project.