**Bilkent University**

**Bilkent 06800, Ankara Turkey**


**CS 461-1 ARTIFICIAL INTELLIGENCE**

**FINAL REPORT - SPRING 2022**

**19.05.2022**

# Reinforcement Learning on Traditional Game "Super Mario"



| **Team Members** | **Student ID** | **Department** |
|---|---|---|
| ● Erdem Deniz Erdağ | 21600951 | Electrical and Electronics Engineering |
| ● Yakup Çatalkaya | 21704019 | Electrical and Electronics Engineering |
| ● Kerem Bayramoğlu | 21801723 | Electrical and Electronics Engineering |
| ● Ata Berk Çakır | 21703127 | Electrical and Electronics Engineering |
| ● Buğra Yiğit Bulat | 21600814 | Electrical and Electronics Engineering |

**Instructor:** Özgur S. Oğuz

**TA:** Sina Barazandeh

**GitHub URL:** https://github.com/erdeme36/Reinforcement-Learning-on-Traditional-Game-Super-Mario

**Table of Content**

## List of Figures

## 1. 0 Abstract

The key challenge that is being focused on is how to play a game optimally according to the game's objectives since the action of playing a game is considered intelligent behavior with reinforcement learning. The issue with the game is that the state and action spaces included in problems related to reinforcement learning are extremely complicated and samples in it are limited. Deep Q Learning facilitates the agent to consider action which is optimal in the provided environment. In Double Q-learning, for an acceleration of the model according to convergence, the state and action space is reduced. The main idea is to compare two theories which are Deep Q Learning and Double Deep Q Learning for the project. As the result, Complex Movement is the winner of Deep Q Learning while Simple Movement for the Double Deep Q Learning.

## 2.0 Introduction

The main problem that is focused on is playing a game optimally according to the objectives of the game because the action called playing a game is considered an intelligent behavior with reinforcement learning. Reinforcement learning is the family of learning algorithms in which an agent learns from its environment. An agent learns to take actions that bring it from its current state to the optimal reachable state. Reinforcement learning provides a unique approach compared to the other machine learning methods. Thanks to our approach, there is artificial interaction with the game environment including trial and error. AI will allow it to give dynamic responses to modifications in the game environment, resulting in a more varied knowledge. Moreover, because of its capability to learn without being managed, it can save development costs by requiring less time to balance and provide solutions for each possible circumstance in which AI might be. An interesting part of the artificial intelligence for this game is that people are very interested in having machines do things they cannot do and want to see the results. It is fascinating for people that artificial intelligence performs these functions perfectly. Although this approach runs perfectly, there are limitations. The shortcomings of reinforcement learning are examined according to the overload and computation. The more this algorithm is used, the more states are overloaded. Moreover, reinforcement learning requires extensive data and computation besides computational power so it is not properly applicable for basic issues. For our problem, a Deep Q learning approach has been decided. This method is not optimal which is illustrated in section 4.0. For further investigation, the Double Deep Q learning method gave better results compared to Deep Q learning. In Deep Q-learning, the action that maximizes the reward is taking the best action in each state. Because the agent has no prior knowledge of the environment, it must first estimate Q values, and then, an update rule is applied. Since the Agent takes the least optimum action in every condition due to having the highest Q-value, the action value is overestimated. While training the model, because Mario needed a lot of episodes to learn the environment and how to act to get more rewards, there was a considerable amount of computational power needed so that Mario's agent could act properly. For that reason, we have spent so much time training the agent. However, in the end, it can be seen that What has been learned from the project is how to use Reinforcement learning to train a model by using trial-error methods. Also, how to use Mario’s pixels after CNN preprocessing as an input to RL has been learned.

## 3.0 Background / Methodology

Under the Background/ Methodology section, firstly the game environment used, preprocessing method and CNN architecture used for the image processing of the environment will be explained. Then two of the algorithms will be inspected under different subsections which are Deep Q Learning and Double Deep Q Learning.

Before starting the concept, since Mario Bros is an environment that depends on reward function, it would be determined for the Q-Learning concept. The main goal is to move Mario as far right as possible and as fast as possible without dying. Mario's environment that was used is SuperMarioBros-1-1-v1. Also, the rewards of Mario Bros depend on three variables will be mentioned.

## 3.1 Problem Domain, Preprocessing, and CNN Architecture

### 3.1.1 Problem Domain

In this section, we will explain the game environment and its features. The game we are trying to play with an AI agent is called SuperMarioBros. In our implementation, we are using version v1. The screenshot of the game environment is given in the figure below.

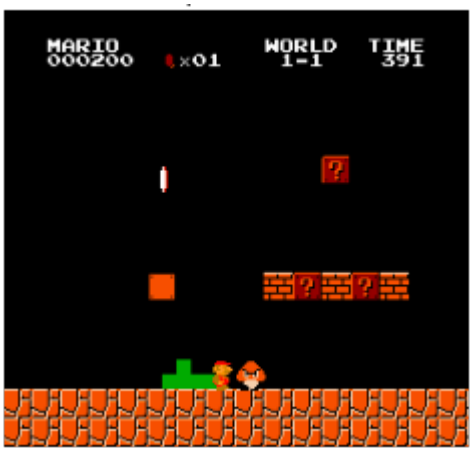


*Figure 1 Super Mario [4]*

To make the environment first we imported gym_super_mario_bros. The environment only sends rewardable gameplay frames to agents. The actions are chosen by the agent according to their rewards. The main goal is to move Mario as far right as possible and as fast as possible without dying. The reward function Mario Bros consists of three different variables which are V, C, and D.

V => It is the instantaneous velocity for the given step which is calculated by subtracting the current position from the position in the previous state. The variable indicates whether Mario is moving right, standing still, or moving left. Since our main goal is to move right, v becomes a positive reward. Oppositely, if Mario moves left which is not the desired direction V becomes a negative reward. In other words:

- moving right $\Leftrightarrow v > 0$
- moving left $\Leftrightarrow v < 0$
- not moving $\Leftrightarrow v = 0$

C => It represents the time passed between the current and previous state. Since one of our goals is to move as fast as possible this variable is necessary for penalizing Mario's standing behavior. For the best case, Mario should always take action.

- not standing $\Leftrightarrow c = 0$
- standing $\Leftrightarrow c < 0$

D => It refers to the death penalty. It penalizes the reward when Mario dies therefore this variable encourages the agent not to die.

- alive $\Leftrightarrow d = 0$
- dead $\Leftrightarrow d = -15$

The total reward is calculated by the following formula;

$$R(total\_reward) = V + C + D$$

Also, the environment returns a dictionary called "info" after each step. This dictionary contains different keys by default and provides different information about the current situation. The keys to the dictionary are given in the following table.

| Key | Type | Description |
|---|---|---|
| `coins` | `int` | The number of collected coins |
| `flag_get` | `bool` | True if Mario reached a flag or ax |
| `life` | `int` | The number of lives left, i.e., *{3, 2, 1}* |
| `score` | `int` | The cumulative in-game score |
| `stage` | `int` | The current stage, i.e., *{1, ..., 4}* |
| `status` | `str` | Mario's status, i.e., *{'small', 'tall', 'fireball'}* |
| `time` | `int` | The time left on the clock |
| `world` | `int` | The current world, i.e., *{1, ..., 8}* |
| `x_pos` | `int` | Mario's *x* position in the stage (from the left) |
| `y_pos` | `int` | Mario's *y* position in the stage (from the bottom) |

*Figure 2 Information Provided by Environment [3]*

### 3.1.2 Preprocessing

The preprocessing phase is important for reducing scalability. Q-Learning has a drawback of stability but it can work with CNN. In this way, some environments, for instance, Mario, can be trained according to the reward. First of all, the Super Mario Bros observation space is 240x256x3 which has 3 channel RGB images. Also, it has 256 action spaces which means that Mario can move 256 different actions at a time. Since, train time would be too much if preprocessing phase had not been done, preprocessing phase is important.

Preprocessing part consists of the following five stages;

1. Max and Skip: This class has two functions which are step and reset. Step function returns “max_frame” (how much distance Mario takes), “total_reward” (calculation explained in the previous part), “done” (contains info whether Mario died or not), and “info”(dictionary explained in the previous part).

```python
class MaxAndSkipEnv(gym.Wrapper):
    """
        Each action of the agent is repeated over skip frames
        return only every `skip`-th frame
    """
    def __init__(self, env=None, skip=4):
        super(MaxAndSkipEnv, self).__init__(env)
        # most recent raw observations (for max pooling across time steps)
        self._obs_buffer = collections.deque(maxlen=2)
        self._skip = skip

    def step(self, action):
        total_reward = 0.0
        done = None
        for _ in range(self._skip):
            obs, reward, done, info = self.env.step(action)
            self._obs_buffer.append(obs)
            total_reward += reward
            if done:
                break
        max_frame = np.max(np.stack(self._obs_buffer), axis=0)
        return max_frame, total_reward, done, info

    def reset(self):
        """Clear past frame buffer and init to first obs"""
        self._obs_buffer.clear()
        obs = self.env.reset()
        self._obs_buffer.append(obs)
        return obs
```

*Figure 3 Max and Skip Environment*

2. Rescale: Since all the images that are taken from Mario are 3 channel RGB 240x256x3, it needs to be converted to grayscale for dimension reduction, and to reduce model complexity since we don't have much computational power. We converted the images to grayscale by using the Luminosity model. The sRGB color space is converted to gray using the following formula

$$Grayscale\ Component\ =\ 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Then images are resized since we reduced the dimension from 3D to 1D. Here is the code that was used.

```python
class MarioRescale84x84(gym.ObservationWrapper):
    """
    Downsamples/Rescales each frame to size 84x84 with greyscale
    """
    def __init__(self, env=None):
        super(MarioRescale84x84, self).__init__(env)
        self.observation_space = gym.spaces.Box(low=0, high=255, shape=(84, 84, 1), dtype=np.uint8)

    def observation(self, obs):
        return MarioRescale84x84.process(obs)

    @staticmethod
    def process(frame):
        if frame.size == 240 * 256 * 3:
            img = np.reshape(frame, [240, 256, 3]).astype(np.float32)
        else:
            assert False, "Unknown resolution."
        # image normalization on RBG
        img = img[:, :, 0] * 0.299 + img[:, :, 1] * 0.587 + img[:, :, 2] * 0.114
        resized_screen = cv2.resize(img, (84, 110), interpolation=cv2.INTER_AREA)
        x_t = resized_screen[18:102, :]
        x_t = np.reshape(x_t, [84, 84, 1])
        return x_t.astype(np.uint8)
```

*Figure 4 Mario Rescale (Gray-Scale 84x84)*

3. Image to Torch: Since the torch library was used for the CNN part, all the images need to be converted to Pytorch tensors.

```
class ImageToPyTorch(gym.ObservationWrapper):
    """
    Each frame is converted to PyTorch tensors
    """
    def __init__(self, env):
        super(ImageToPyTorch, self).__init__(env)
        old_shape = self.observation_space.shape
        self.observation_space = gym.spaces.Box(low=0.0, high=1.0, shape=(old_shape[-1], old_shape[0], old_shape[1]), dtype=np.float32)

    def observation(self, observation):
        return np.moveaxis(observation, 2, 0)
```

*Figure 5 Convert Array of Images to Pytorch Arrays*

4. Buffer Wrapper: Since Q-Learning is a discrete method, discrete values need to be collected at each time step in the buffer.

```
class BufferWrapper(gym.ObservationWrapper):
    """
    Only every k-th frame is collected by the buffer
    """
    def __init__(self, env, n_steps, dtype=np.float32):
        super(BufferWrapper, self).__init__(env)
        self.dtype = dtype
        old_space = env.observation_space
        self.observation_space = gym.spaces.Box(old_space.low.repeat(n_steps, axis=0),
                                                old_space.high.repeat(n_steps, axis=0), dtype=dtype)

    def reset(self):
        self.buffer = np.zeros_like(self.observation_space.low, dtype=self.dtype)
        return self.observation(self.env.reset())

    def observation(self, observation):
        self.buffer[:-1] = self.buffer[1:]
        self.buffer[-1] = observation
        return self.buffer
```

*Figure 6 Buffer Wrapper*

5. Pixel Normalization: Converted images are 256 pixels starting from 1. Thus, it should be divided by 255 to be normalized.

```
class PixelNormalization(gym.ObservationWrapper):
    """
    Normalize pixel values in frame --> 0 to 1
    """
    def observation(self, obs):
        return np.array(obs).astype(np.float32) / 255.0
```

*Figure 7 Pixel Normalization*

By implementing all these steps accordingly, a functional Mario environment can be obtained to use for Deep Q-Learning.

### 3.1.3 CNN (Convolutional Neural Network)

For this project, CNN was used to image processing for Mario. Because CNN gives high accuracy and can be used with Q-Learning, it gives better scores for the deep-learning architectures. The model was constructed as an input layer and output layer. Here is the model that was used;

```
Sequential(
  (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU()
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU()
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU()
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU()
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU()
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Flatten(start_dim=1, end_dim=-1)
  (16): Linear(in_features=4096, out_features=1024, bias=True)
  (17): ReLU()
  (18): Linear(in_features=1024, out_features=512, bias=True)
  (19): ReLU()
  (20): Linear(in_features=512, out_features=10, bias=True)
)
```

*Figure 8 CNN Structure with 3 Convolutional Layers and 2 Linear Layers*

In the Convolution Neural Network, Convolutional layers, Linearization Layers, and Activation functions were used. The network is divided into two parts: the main layer and the output layer. The convolutional neural network is made up of numerous layers of neurons, each of which has a nonlinear operation on the outputs of the layer. Convolutional and pooling layers are the most common layers of CNN. By learning from the images, it updates weights according to backpropagation and optimizes gradients by using one of the optimizers (AdaGrad, RMSProp, and Adam).

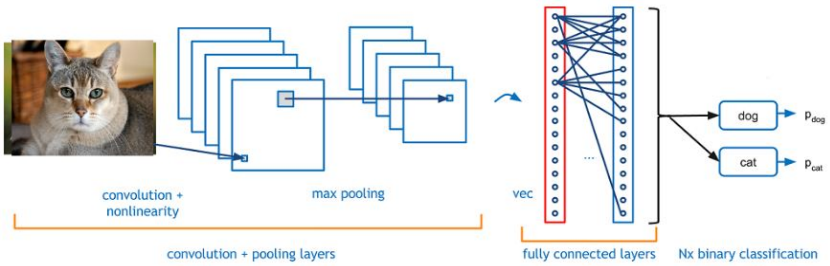


*Figure 9 Architecture of CNN [5]*

Convolutional Layers: The main concept of convolutional layers is to capture low-level features of images and extract high-level features to the next layer. By doing that use Max-Pooling according to kernel size and stride.

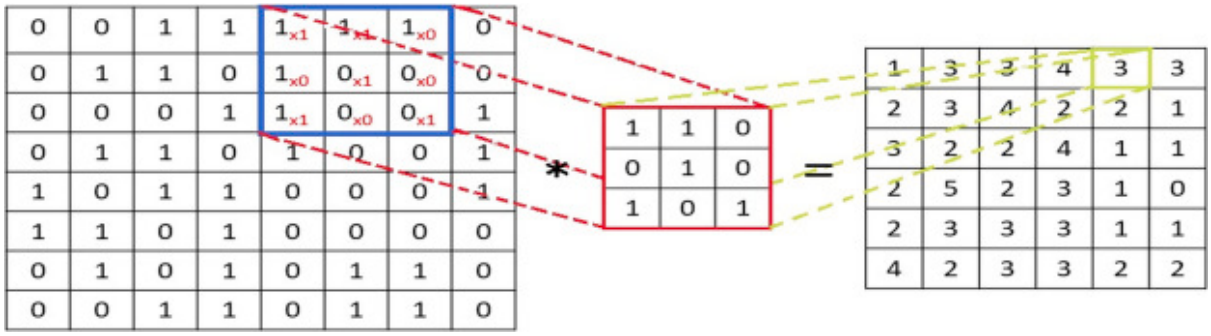

*Figure 10 Max-Pooling in Convolutional Layers [4]*

Adam optimizer was used as an optimizer since it is an improved version of AdaGrad and RMSprop. Adam solves two problems;

1. Accumulation of Gradients (RMSprop)
2. Decaying Problem (AdaGrad)

Update Rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

*Figure 11 Update Rule for Adam [6]*

Rectified Linear Activation function (ReLU) is a partial linear function that, if its value is positive, will output the same input directly; otherwise, it will output zero. It's commonly used in neural networks since it's easier to train and gives better outcomes. The graph of the ReLu function is as follows:

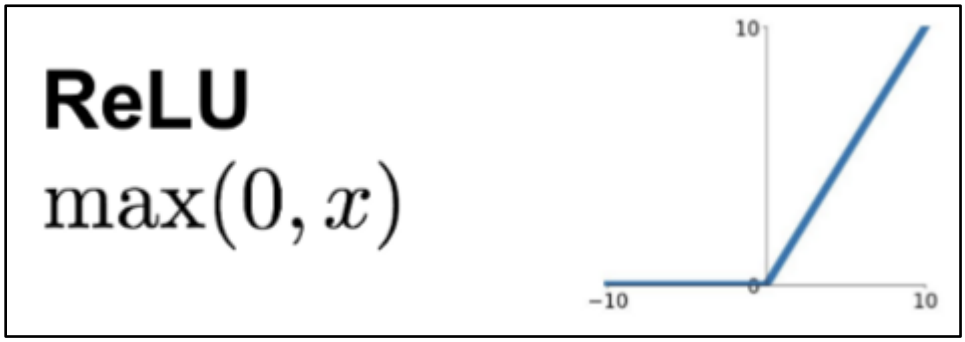


*Figure 12 Relu Activation Function*

### 3.2 Deep Q-Learning

The Deep Q-Learning algorithm is one of the most widely used concepts in Reinforcement Learning and is an appropriate method for the environment. Therefore, we have decided to move on with this algorithm. Because Mario can go to any position in the environment that is nearby the current position, we have a horizontal, vertical, and diagonal set of actions for every step. A reward function defines how well one is in a certain state while performing a specific action. Besides the action in the final step, all other actions in different steps must have a reward. These specific rewards belonging to different actions are denoted as Q(s, a) which

represent the quality (reward) of these specific actions. Then all those Q values are stored in a table called Q-table. When Mario decides its next move, it uses this table and chooses the action that has the highest reward after learning from different trials. In other words, it will take the action where $argmax_a Q(s, a)$. Q-learning is a sort of temporal difference learning that employs a version of the Bellman-update equation. The update equation for the Q learning is the following:

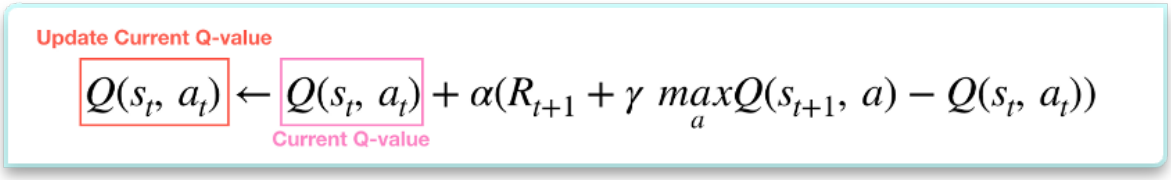


*Figure 13 Update Rule of Q-value*

The reward of being in a specific state $s_t$ and performing an action $a_t$ is determined not only by the reward taken from that specific action but also by the best next move Mario can make in a state $s_{t+1}$ after reaching it. The parameter $\gamma$ is known as the discount factor which is a number between 0 and 1. It determines the significance of the future states in the update procedure. Lastly, $\alpha$ is the learning rate that determines the step size of updating Q-values. The algorithm is very simple and described in the diagram given below.

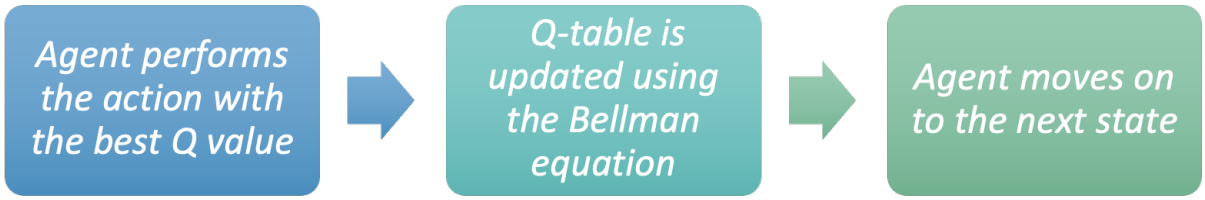


*Figure 14 Overall Algorithm Structure*

In Deep Q-Learning, a neural network substitutes the typical Q-table. Instead of translating a state-action pair to a q-value, a neural network maps input states to (action, Q-value) pairings. Pseudocode for deep Q-Learning algorithm is given below.

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
**for** episode $= 1, M$ **do**
  Initialise state $s_t$
  **for** $t = 1, T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action $a_t$ and observe reward $r_t$ and state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
    Set $s_{t+1} = s_t$
    Sample random minibatch of transitions $(s_t, a_t, r_t, s_{t+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$
  **end for**
**end for**

*Figure 15 Pseudo-Code of Deep Q-Learning [2]*

### 3.3 Double Deep Q-Learning

It must be addressed that one important problem of Q-learning is overestimation bias, that is meaning the Q-values learned are greater than they need to be. The $\max_a Q(s_{t+1}, a)$ comes close to $E(\max_a Q(s_{t+1}, a))$, which is greater than $\max_a(E(Q(s_{t+1}, a))$, the correct Q-value. We employed a technique known as double Q-learning to obtain more precise Q-values. We have two Q-tables in double Q-learning: one for taking actions and another for the Q-update equation particularly.

**Double Q-Learning**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma\, Q'(s_{t+1}, a) - Q(s_t, a_t))$$

estimated/expected Q-value

$$a = \max_a Q(s_{t+1}, a)$$

$$q_{estimated} = Q'(s_{t+1}, a)$$

*Figure 16 Double Q learning Formula*

**Algorithm 1** Double Q-learning

Initialize $Q^A$,$Q^B$,$s$
**repeat**
  Choose $a$, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe $r$, $s'$
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  **if** UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  **else if** UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  **end if**
  $s \leftarrow s'$
**until** end

*Figure 17 Double Q learning Pseudocode*

Q and Q' are two separate action-value functions used in Double Q-Learning. The updating method differs somewhat from the standard version. The Q function is used to choose the optimal action with the highest Q-value of the following state. The Q' function is used to calculate the expected Q-value using the action A.

## 4.0 Results / Evaluations

### 4.1 Deep Q Learning

For the result part, Super Mario has been trained in 1000 samples where the exploration rate is 0.95. It means that randomness was high to teach the environment to Mario quickly. Before starting, the main point is that there are two algorithms and three actions. Therefore, all combinations of each have been trained and tested. Note that for a more clear result, the training part should be extended more. Because of the computational power issue, six experiments trained up to 2000 epochs each.

### 4.1.1 Deep Q Learning for Right Only Action

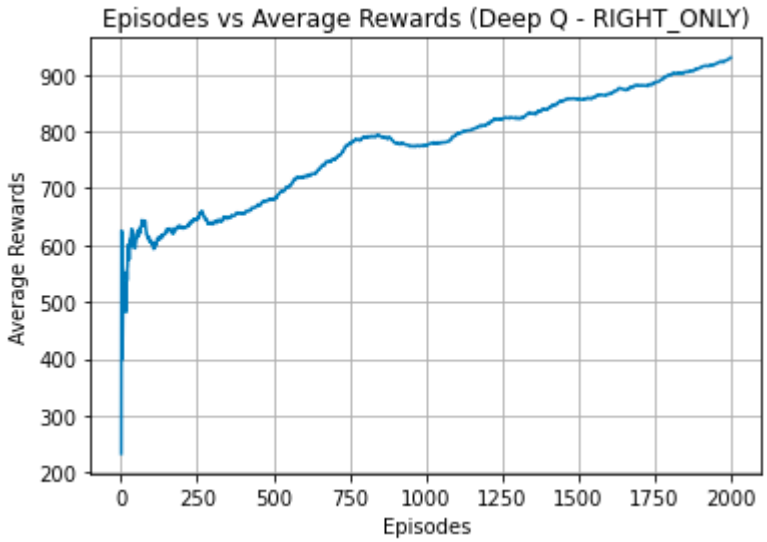


*Figure 18 Average Rewards for Right Only (Deep Q-Learning)*

Deep Q is the first algorithm of the Super Mario Bros project. It can be seen that the average score reached 930.11. In terms of fast convergence, it decays quickly. However, this algorithm is not enough to converge fast compared to Double Deep Q - Learning. It was expected that Mario would reach higher grades quickly by using the Double Deep Q - Learning Algorithm.

### 4.1.2 Deep Q Learning for Simple Movement

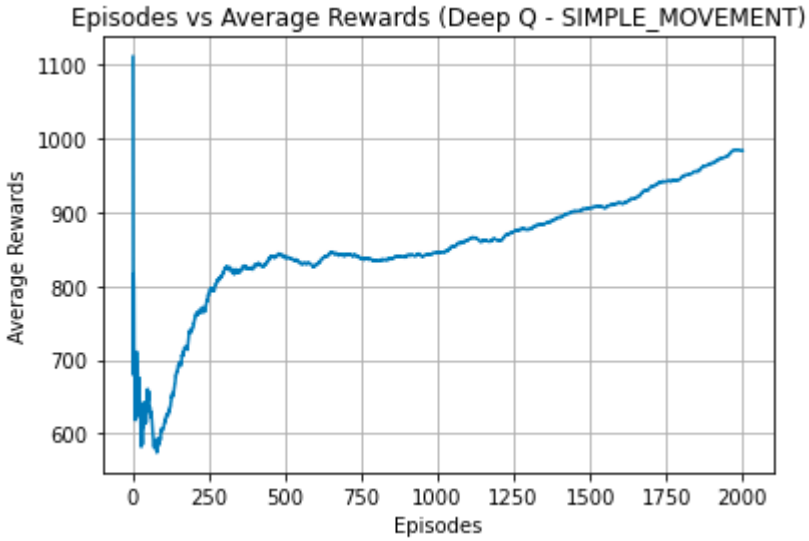


*Figure 19 Average Rewards for Simple Movement (Deep Q-Learning)*

This graph shows the Deep Q learning algorithm implementation by using simple movement action space. It can be seen that the average score reached 983.70. In terms of fast convergence, it is slow compared to complex and right-only movements. Similarly, this algorithm is not enough to converge fast compared to Double Deep Q - Learning.

### 4.1.3 Deep Q Learning for Complex Movement

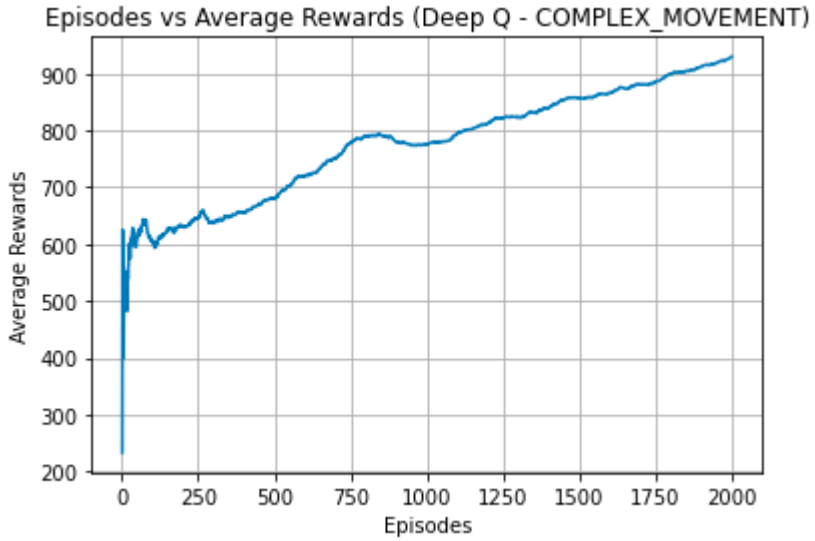


*Figure 20 Average Rewards for Complex Movement (Deep Q-Learning)*

It can be seen that the average score reached 1173.62. Complex movement decays slowly since instead of the other two actions, it has more action. Thus, this time Mario tried to learn algorithms by acting randomly. Since the aim of the game is to go right as much as possible without dying, going left does not return any reward. However, because Mario dies when touching the enemies, by taking action differently may rescue Mario. As a result, Mario can get better scores by taking complex movements.

## 4.2 Double Deep Q Learning

### 4.2.1 Double Deep Q Learning for Right Only Action

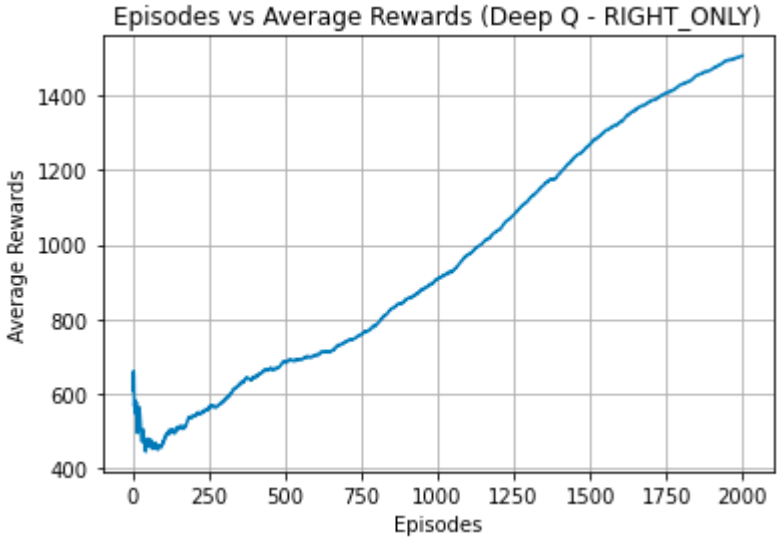


*Figure 21 Average Rewards for Right Only (Double Deep Q-Learning)*

It can be seen that the average score reached 1506.40. It was expected that Mario would reach higher grades quickly since the best gameplay depends on the right actions. The average rewards are increasing linearly while episodes are increasing. So it can be said that double q values are getting more optimal with higher average rewards.

### 4.2.2 Double Deep Q Learning for Simple Movement

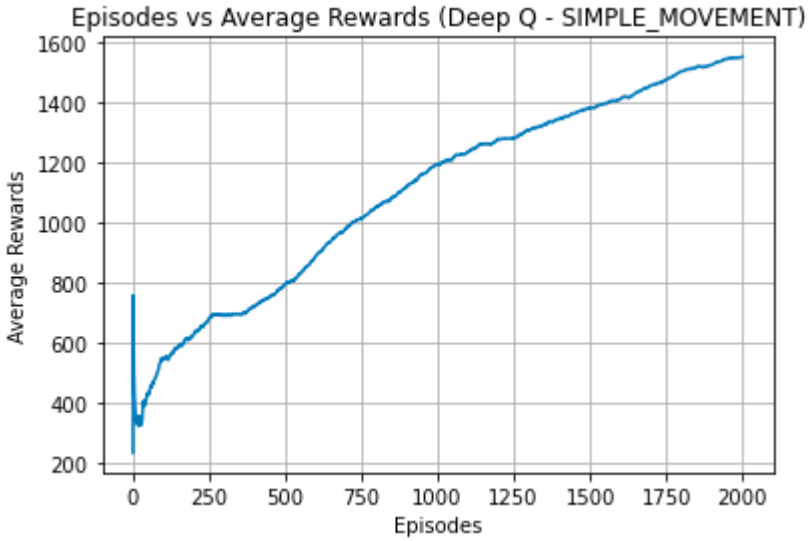


*Figure 22 Average Rewards for Simple Movement (Double Deep Q-Learning)*

It can be seen that the average score reached 1553.90. It was expected that Mario would reach higher grades quickly. The average rewards are increasing linearly while episodes are increasing. So, it can be said that double q values are getting more optimal with higher average rewards. Compared to double deep q learning with complex movements, the algorithm is reaching higher rewards with simple movements.

### 4.2.3 Double Deep Q Learning for Complex Movement

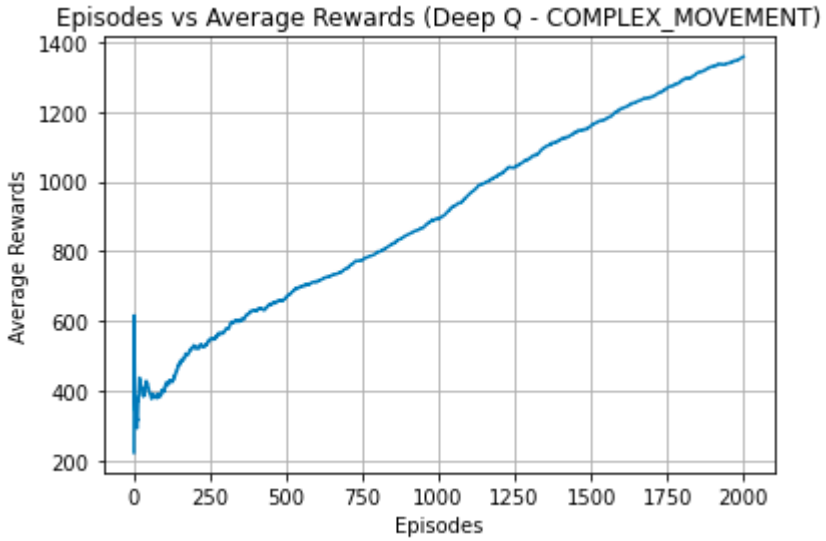


*Figure 23 Average Rewards for Complex Movement (Double Deep Q-Learning)*

It can be seen that the average score reached 1357.61. It was expected that Mario would reach higher grades quickly. The average rewards are increasing linearly while episodes are increasing. So it can be said that double q values are getting more optimal with higher average rewards. Compared to double deep q learning with simple movement and right only action, the algorithm is reaching lower rewards with complex movements.

### 4.3 Comparison of Two Algorithms with Three Actions

#### 4.3.1 Comparison of Two Algorithms for Right Only

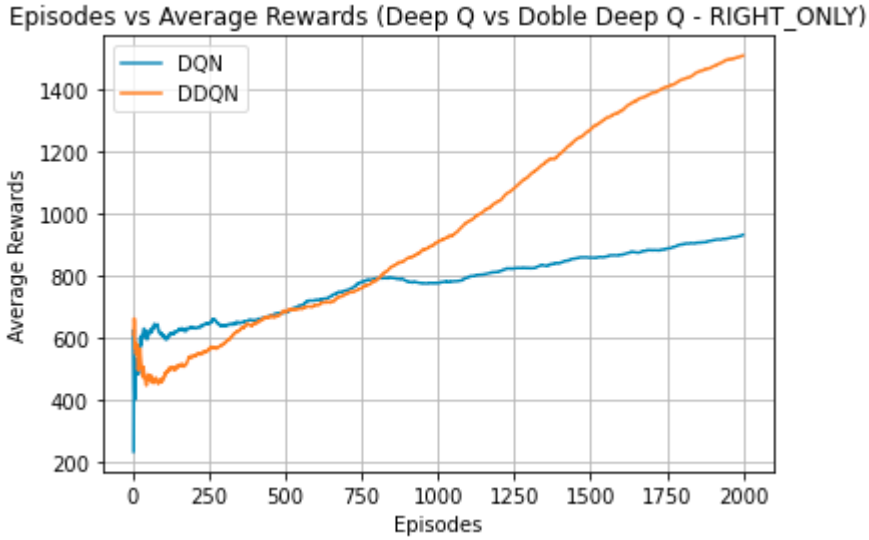


*Figure 24 Average Rewards for Complex Movement*

It can be easily said that double deep q-learning gives much better scores than deep q learning. With the right only action, the double deep q learning algorithm is starting slow compared to deep q learning but double deep q learning is reaching considerably more rewards compared to deep q learning because of more optimal q values in double deep q learning. The difference between Deep Q and Double Deep Q is 576.29.

#### 4.3.2 Comparison of Two Algorithms for Simple Movement

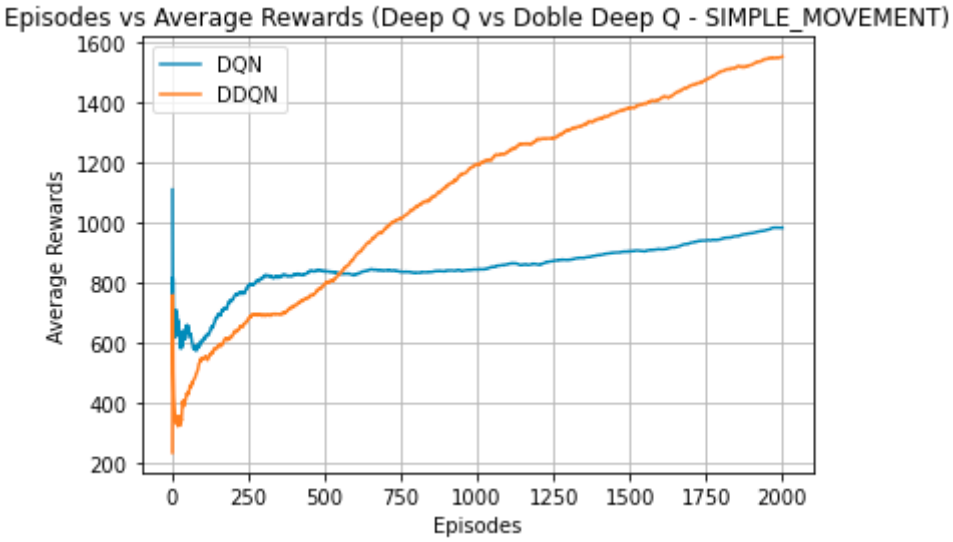


*Figure 25 Average Rewards for Complex Movement*

With a simple movement, the double deep q learning algorithm is starting slow compared to deep q learning but double deep q learning is reaching considerably more rewards compared to deep q learning because of more optimal q values in double deep q learning. By using the Deep q learning algorithm, it can be seen that it converges very fast. After the 750 epochs, the reward increases slowly. However, with double deep q learning, it did not converge and increased quickly. The difference between Deep Q and Double Deep Q is 570.2.

### 4.3.3 Comparison of Two Algorithms for Complex Movement

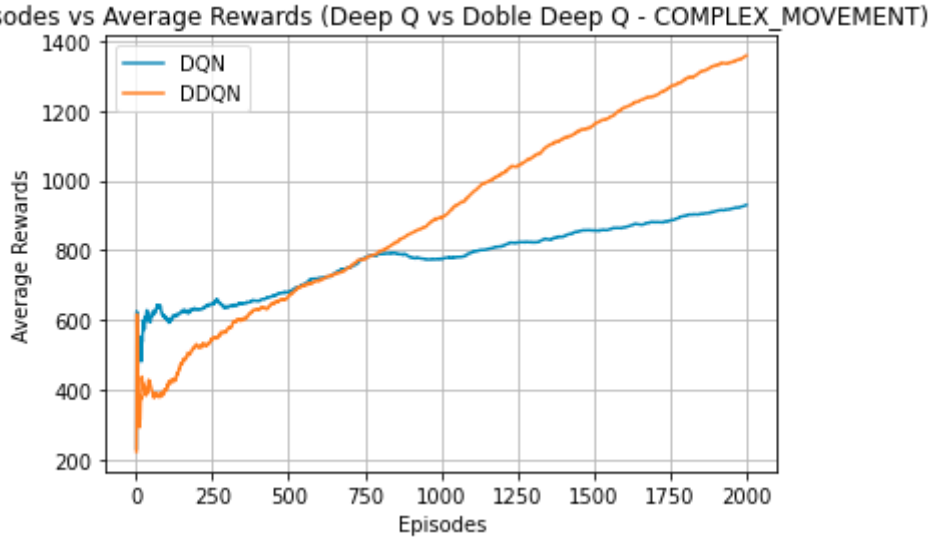


*Figure 26 Average Rewards for Complex Movement*

With complex movement, the double deep q learning algorithm is starting slow compared to deep q learning but double deep q learning is reaching considerably more rewards compared to deep q learning because of more optimal q values in double deep q learning. In the complex case, both algorithms did not converge. They act similarly but double deep q learning decays fast. The difference between Deep Q and Double Deep Q is 183.99.

### 4.3.4 Comparison of Three Action for Deep Q Algorithm

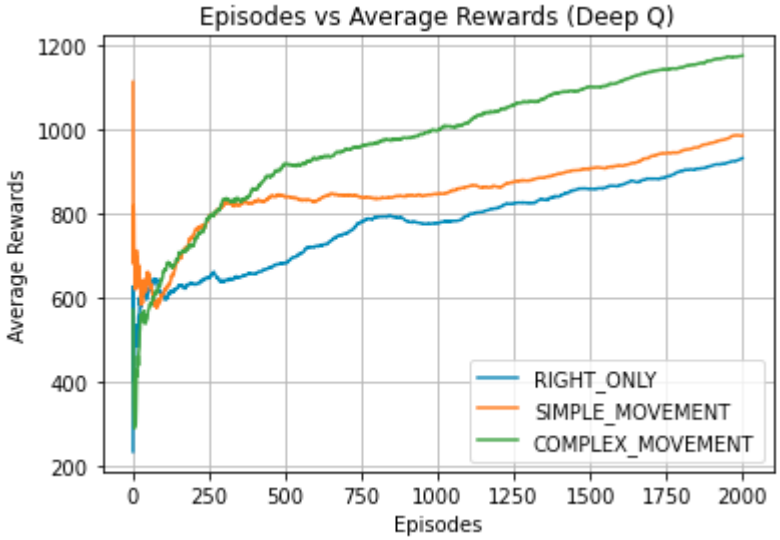


*Figure 27 Average Rewards of Deep Q-Learning for Each Action*

This graph shows the comparison of three actions for Deep Q-Learning Algorithms. As can be seen, Complex Movement is the best action compared to Right Only and Simple Movement. While considering deep q learning algorithms, complex movement is getting the highest rewards even though there is a peak at the beginning for simple movement during episodes. The lowest average reward is an algorithm with the right only action. Mainly, the best-expected reward should belong to the Right Only movement since Mario needs to be allowed to move right as much as possible.

### 4.3.5 Comparison of Two Algorithms for Double Deep Q Algorithm

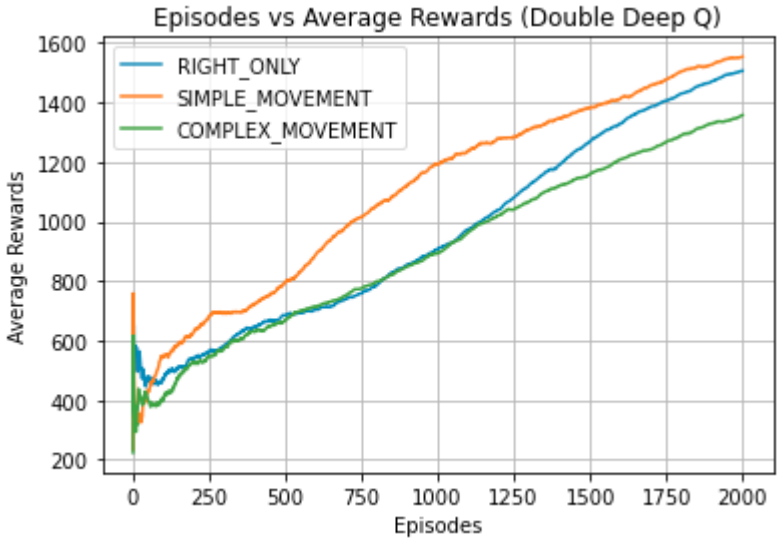


*Figure 28 Average Rewards for Complex Movement*

This graph shows the comparison of three actions for the Double Deep Q-Learning Algorithm. The one thing that was observed before is that Double Deep Q-Learning gets better results compared to Deep Q-Learning since Double Deep Q-Learning uses two different action-value functions (Q and Q'). As can be seen, Complex Movement is the best action compared to Right Only and Simple Movement. The lowest average reward is an algorithm with a complex movement as expected.

**5.0 Discussions**

Deep Q-Learning and Double Deep Q-Learning were implemented and tested up to 2000 epochs for each action. The disadvantage of Q-Learning is that it is a reinforcement algorithm that determines the maximum predicted reward. If an action value was overestimated in this scenario, it was automatically picked. However, evaluating both Q values results in higher marks for determining the optimal action.

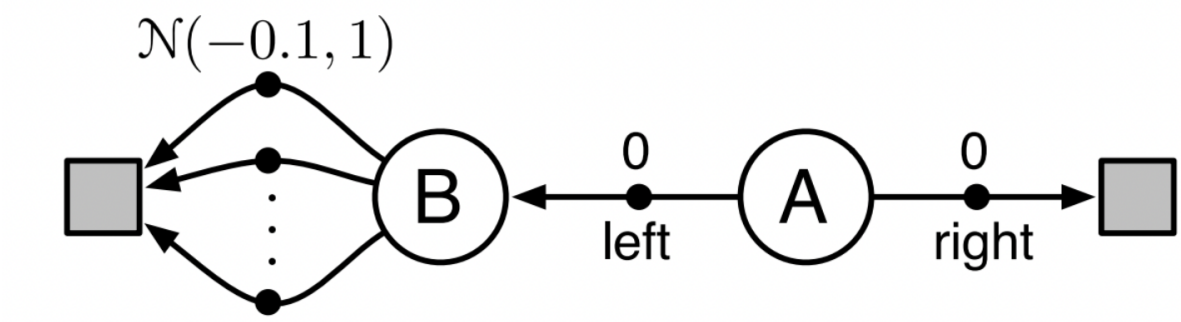


*Figure 29 Visualizing Which Node Direction is Better [9]*

In this example, if the current state is A and the left Q value is greater than the right, the agent will leave. It is unaware, however, that all incentives are detrimental in this regard. The experimental findings of Q-Learning and Double Q-Learning are shown in the graph below.

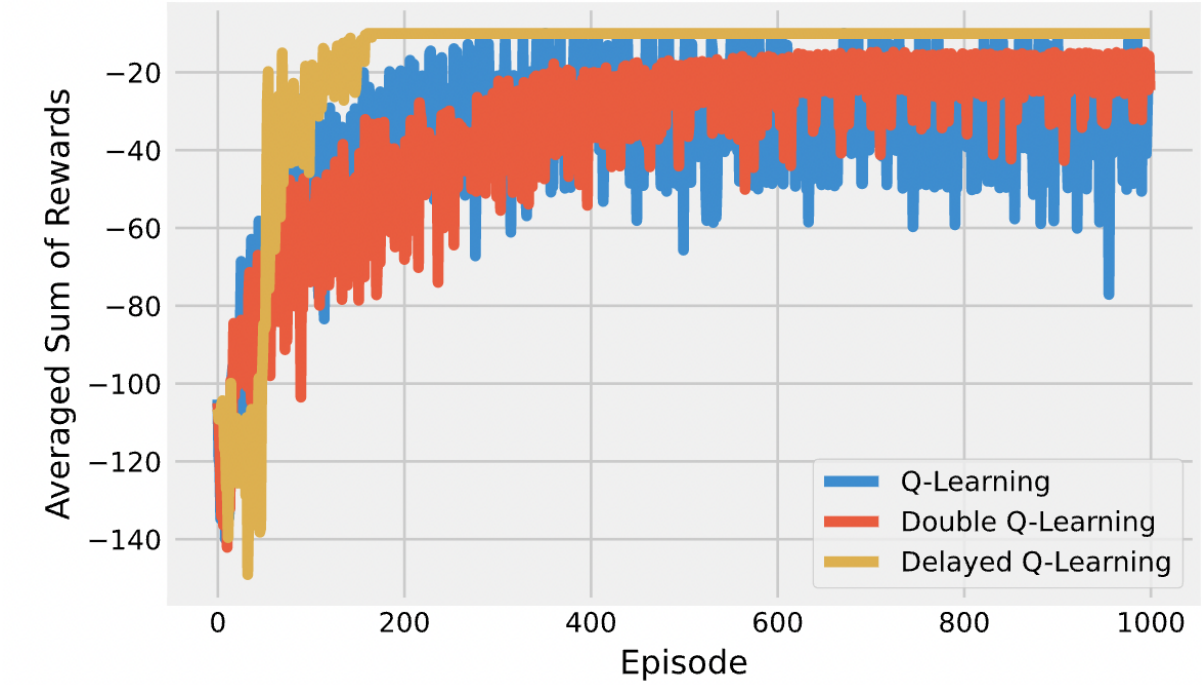


*Figure 30 Comparisons for Q-Learning and Double Q-Learning [9]*

To reduce the overestimation caused by maximizing, DDQN utilizes two Q-networks, one for getting actions and the other for updating weights through backpropagation. Overestimation difficulties can be mitigated in this fashion.

### 5.1 Further Improvement

In this section, the main goal is to research how this project can be improved in such a way that training rewards get higher and Mario moves right as much as possible without dying.
In the feature extraction part, CNN was used. However, if RNN or LSTM which stores previous actions was used, it gives better results. Also, for this project, there are different Super Mario Bros environments. These environments need to be implemented and trained with this methodology.

| Environment | Game | ROM |
|---|---|---|
| SuperMarioBros-v0 | SMB | Standard |
| SuperMarioBros-v1 | SMB | Down Sample |
| SuperMarioBros-v2 | SMB | Pixel |
| SuperMarioBros-v3 | SMB | Rectangle |
| SuperMarioBros2-v0 | SMB2 | Standard |
| SuperMarioBros2-v1 | SMB2 | Down Sample |

*Figure 31 Environments for Super Mario Bros [3]*

## 6.0 References

[1] “What makes ReLU so much better than Linear Activation? As half of them are exactly the same,” Quora. [Online]. Available: https://www.quora.com/What-makes-ReLU-so-much-better-than-Linear-Activation-As-half-of-them-are-exactly-the-same. [Accessed: 30-Apr-2022].

[2] Researchgate.net. [Online]. Available: https://www.researchgate.net/figure/Pseudo-code-of-deep-Q-learning-with-experience-replay_fig3_338776794. [Accessed: 03-May-2022].

[3] “Gym-super-mario-bros,” PyPI. [Online]. Available: https://pypi.org/project/gym-super-mario-bros/. [Accessed: 03-May-2022].

[4] G. Carneiro, J. Nascimento, and A. P. Bradley, “Deep Learning Models for Classifying Mammogram Exams Containing Unregistered Multi-View Images and Segmentation Maps of Lesions11This work is an extension of the paper published by the same authors at the Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015) [1],” in Deep Learning for Medical Image Analysis, Elsevier, 2017, pp. 321–339.

[5] GeeksforGeeks. 2021. Project Idea | Cat vs Dog Image Classifier using CNN implemented using Keras-GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/project-idea-cat-vs-dog-image-classifier-using-cnn-implemented-using-keras/> [Accessed 26 December 2021].

[6] Medium. 2021. Learning Parameters Part 5: AdaGrad, RMSProp, and Adam. [online] Available at: <https://towardsdatascience.com/learning-parameters-part-5-65a2f3583f7d> [Accessed 26 December 2021].

[7] *PPO*. PPO - Stable Baselines3 1.5.1a0 documentation. (n.d.). Retrieved March 31, 2022, from https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017, August 28). *Proximal policy optimization algorithms*. arXiv.org. Retrieved May 2, 2022, from https://arxiv.org/abs/1707.06347

[9] A. Deshpande, “Deep Double Q-Learning — Why you should use it,” Medium, 26-May-2018.[Online].Available: https://medium.com/@ameetsd97/deep-double-q-learning-why-you-should-use-it-bedf660d5295. [Accessed: 03-May-2022].