Machine Learning for Game Playing

Student Name: Muhamad Fahim Aqil Bin Muhamad Sahlan
Supervisor Name: Suncica Hadzidedic

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Science, Durham University.

Abstract –
Context: Super Mario Bros is a popular action game with a "real-life" environment and large state space. Super Mario Bros can be used as a platform to train a computer player that self-play the game using Reinforcement Learning. The computer player requires to interact with various objects and obstacles in the Super Mario Bros environment, encouraging a knowledge-rich approach in learning process.

Aims: The aim of this project is to implement a state-of-the-art Deep Reinforcement Learning approach which is Proximal Policy Optimization to train an agent to self-play Super Mario Bros. Subsequently, we implemented a new variation of the approach and assessed the performance of our method with the general Proximal Policy Optimization implementation.

Method: We used the framework provided by OpenAI Gym - Super Mario Bros gym - and extracted the information from the game environment to train an RL agent using Proximal Policy Optimization. We also introduced preprocessing methods such as frame scaling, stochastic frameskip, frame stacking and, noisy net to the environment to improve the performance of our agent. A variation of Proximal Policy Optimization is created by introducing a rollback operation to improve the stability of the training.

Results: Our approach managed to train an agent that was able to beat the first level of World 1 within 20 hours of training time. We successfully implemented a method that can perform better than the general Proximal Policy Optimization implementation with a 50% increase in performance, without data preprocessing, and 10 % increase with data preprocessing.

Conclusion: The solution improves the current state of the art by implementing a rollback function and adequate data preprocessing. The statement is supported by a clear description of how our system is evaluated.

Keywords **–** Reinforcement Learning, Computer player, Super Mario Bros agent, OpenAI Gym

I. INTRODUCTION

Machine Learning is a widely spoken topic nowadays. One of the most interesting areas of machine learning is Reinforcement Learning (RL). RL is one of the branches of Machine Learning which involves a learning agent interacting with its environment to achieve goals. The main focus of RL is to maximize a reward signal by learning from its own experience. Furthermore, the process of RL involves trial and error interaction with the environment (Sutton and Barto, 1998: 2).

RL has been widely used in various fields such as order identifying the best treatment for patients in healthcare (Gottesman et al., 2019) and helping investors to make a saving decision in finance (Choi et al., 2009). However, prior works in RL are usually concentrated on video and computer games due to the breakthrough of AlphaGo (Silver et al., 2016) by DeepMind in defeating a World Go Champion player without any handicap. Another notable achievement in RL is the emerging of AI players that able to defeat Dota 2 professional players in five versus five matches (Berner et al., 2019). Moreover, the interest of conducting research on RL increases since the introduction of interfaces that aids the process of implementing RL such as OpenAI Gym (Brockman et al., 2018) and Atari Learning Environment (Bellemare et al., 2013).

The main focus of this project is to implement the state-of-the-art RL algorithm, which is Proximal Policy Optimization (PPO) (Schulman et al., 2017) to train a computer player to complete the level of Super Mario Bros (SMB). We demonstrated by integrating a new variation of the PPO, and the implementation of data preprocessing to the SMB environment, our solution was able to improve the performance of the agent.

A. Reinforcement Learning and Markov Decision Process (MDP)

One of the exciting challenges for RL to solve is to construct a mathematical framework for MDP. MDP is a way to formalize the sequential decision process in the stochastic environment, and it consists of 4 components; decision epochs, states, rewards and transition probabilities (Sutton and Barto, 1998:53). The model requires to choose an action through transition probabilities. Each action generates a reward to determine the state at the next decision epoch. An optimal policy must be computed as a guide for the decision-maker to choose the best action at every future decision epoch—however, the process of acquiring the optimal policy is difficult due to the stochasticity of the environment.

RL is ideal as an approach to solve the problem mentioned above since it is identified through the four main sub-elements: a policy, a reward signal, a value function, and a model of the environment. The policy controls the way a learning agent to behave in a given time. It maps the states of the environment to the potential actions taken. A reward signal is the main goal in an RL problem. A reward can be acquired at each timestep from the environment, and the agent requires to maximize the accumulative rewards over the long run. The reward is also an indicator for the agent to differentiate between good and bad events. A value function is used to specify what good in the long run. It can provide the agent with a prediction of the total amount of reward that can be accumulated over the future. The last one, which is a model of the environment, allows inferences of the environment to behave.

B. Proximal Policy Optimization

The motivation to apply PPO in solving multiple problems arise from the recent surge popularity and effectiveness of PPO in achieving state-of-the-art performance in various RL benchmarks (Schulman et al., 2017). PPO is a family of policy gradient methods that use multiple stochastic gradient ascent to perform each policy update. PPO objective is to compute an estimator of the policy gradient and plugging it into a stochastic gradient ascent. In order to prevent the large policy updates existed in policy gradient, it introduces a clipping function to improve training stability by limiting change to the policy at each step. PPO calculates the ratio between the current policy with the old policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta old}(a_t \mid s_t)} \tag{1}$$

where π_θ is a stochastic policy of the set of actions a given states s at timestep t. PPO takes inspiration from Trust Region Policy Optimization (TRPO) by Schulman (2015) which maximizes a "surrogate" objective function:

$$L^{CPI}(\theta) = \mathbb{E}_t[\, r_t(\theta) A_t] \tag{2}$$

where L represents loss, CPI represents Conservative Policy Iteration, A represents the advantage from GAE and $\mathbb{E}$ is the empirical average over a finite batch of the sample, t represents timestep.

Since TRPO has constraints that limited its function, a Clipped Surrogate Function to find the loss function is created:

$$L^{CLIP}(\theta) = \mathbb{E}_t\, [\, \min(r_t(\theta)\, A_t\, , clip(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t)\,] \tag{3}$$

where $\epsilon = 0.2$, is a hyperparameter. The first minimum term in Eq. (3) is L^{CPI} which can be defined from Eq. (2), the second minimum term modifies the surrogate by clipping the $r_t(\theta)$ which makes sure the ratio is within the range $[1-\epsilon\ ,\ 1+\epsilon\,]$ hence stabilizing the policy. The authors made a comparison between the performance of PPO and other RL algorithms on the Atari domain with 49 games included in OpenAI Gym at the time of publication. It outperformed other algorithms by winning the majority of the games with the highest score.

C. Super Mario Bros Environment

One of the games available through OpenAI Gym, Super Mario Bros (SMB), is a popular two-dimensional platform game that requires the player to control a character named Mario and complete the game by beating all the levels. Player has to navigate Mario to go through all levels by killing enemies, jumping over obstacles and preventing Mario from losing all his lives. SMB is an interesting domain for RL since it has a large state space but with simple dynamics. The various objects existed in the SMB are ideal for encouraging the RL agent to learn by interacting with the objects in order to achieve its goal. SMB is a suitable platform to train an RL agent since it can be formulated as a Markov Decision Process:

- States: A sequence of frames, where each frame is a matrix constructed from the tiled interface from the OpenAI Gym emulator.
- Actions: A sequence of heuristic-based action combinations across multiple frames
- Rewards: Moving right will return a positive reward to the agent, moving left will return a negative reward, a death penalty with a negative reward for dying and negative reward as the agent takes more time to complete the level.

D. Project Objectives

The proposed research questions are: Can the state-of-the-art agent; PPO used to train an RL agent that can self-play levels of SMB and what is the improvement that can be applied to improve the state-of-the-art implementation? The projects were divided into three levels of objectives: minimum, intermediate and advanced.

One of the minimum objectives of this project was to train a computer player that can self-play to complete the first level of World 1 in SMB. The computer player can be trained by implementing the state-of-the-art algorithm, PPO. Schulman et al. (2017) demonstrated in the literature that PPO performed well compared to difference RL algorithm in solving RL tasks. Another minimum objective was to improve the implementation of an agent trained with PPO by integrating data processing from Mnih et al. (2013), Fortunato et al. (2019) and Nichol et al. (2018). A rollback function also included with the PPO to further improve the general PPO implementation (Wang et al., 2019). Through these objectives, a strong understanding of PPO architecture can be required. Furthermore, a foundational model can also be built for the workflow of the project.

The intermediate objective was to create an agent that can self-play all 32 levels in 8 worlds of SMB. Nichol et al. (2018) trained PPO agents on each of the levels in Sonic The Hedgehog to create computer players that can play all levels of the game. Our PPO implementation was also expected to achieve the same achievement as the Sonic agent since the problem dimension of both games is similar. The purpose of this objective was to evaluate the ability of the PPO implementation to solve tasks in different levels of SMB.

The advanced objectives were to implement a meta-learning algorithm for the implementation. Meta-learning can speed up the process of PPO training by introducing a generalized policy to the PPO. In order to create a meta learner (Nichol et al., 2018), high computational resources must be used since the method trained parallel agents on multiple levels at the same time. The motivation of this purpose was to extend the current approach to another level by implementing a meta learner, which is useful in improving any RL algorithms. (Nichol et al., 2018).

II. RELATED WORK

Significant academic works have been conducted using the different RL algorithms to games with similar dimensions as an SMB environment. In this section, an overview of different RL algorithms implemented in different games is given. Prior works on implementing RL algorithms to Super Mario Bros are also discussed in this section.

A. Implementation of Reinforcement Learning in Different Games

One of the approaches in implementing the RL algorithm is by using a Q-Learning approach. Q-Learning is different from a policy gradient approach since rather than mapping the states to its action using a policy, Q-Learning learns a single deterministic action from a discrete set of actions by finding the maximum value. Mnih et al. (2013) invented an algorithm named Deep Q Network (DQN), which used a convolutional neural network, trained with a variant of Q-learning. An RL ATARI 2600 agent was trained, and it achieved high scores that can match the achievement of expert human players.

Mnih et al. (2016) introduced a classic version of policy gradient method, which is Asynchronous Actor-Critic (A3C). Actor-critic used learned value function (Q-learning) in order to assist the policy update. The authors focused on parallel training while learning the best policy. The algorithm was implemented to Atari 2600 games, and it outperformed the scores gained by DQN. Wu and Tian (2017) used actor-critic and curriculum training to train an agent playing a First-Person Shooter (FPS) game. The method successfully won 10 out the 11 attended games.

Schulman et al. (2015) introduced Trust Region Policy Optimization (TRPO), which applied KL Divergence constraint on the size of policy update at each iteration. The algorithm was tested with Atari 2600 games and simulated robotic tasks. It successfully produced better outcomes than DQN and human players. TRPO and actor-critic methods are the main foundation of PPO architecture. Schulman et al. (2017) used both of the approaches to create PPO, thus producing the state-of-the-art algorithm that performed better than both algorithms.

Nichol et al. (2018) conducted a benchmark for generalization in RL by implementing various state-of-the-art algorithms with the meta-learning algorithm. The authors implemented the algorithms to the Sonic The Hedgehog game. One of the algorithms implemented in the benchmark, which was Joint PPO performed the best by achieving the highest average scores for all the levels in Sonic The Hedgehog.

B. Implementation of Reinforcement Learning in Super Mario Bros

Liao and Yang (2012) used the Q-Learning approach in SMB by using an epsilon-greedy exploration method. The author introduced four metrics to evaluate the performance which is the total reward gained by the agent given certain iterations, the probability the agent beats the level the percentage of monster killed, and the time spent on the level. By introducing a reward function for killing an enemy, the agent managed to train an agent that has a high killing rate and consistently beats the first level of World 1. The authors demonstrated by specifying reward functions to do specific tasks, the agent can learn to new task such as killing enemies.

Grand and Loughlin (2018) applied three RL algorithms; Q-learning, Approximate Q and SARSA in SMB and compared the performance of the algorithms with a heuristic agent. The SMB agent required to complete the first level of World 1. The heuristic agent was hardcoded to get the agent to avoid gaps and enemies. The RL agents were unable to beat the performance of the heuristic agent. However, the most valuable finding of this work was that the authors introduced a performance matrix by using mean and maximum distance achieved by the agent given certain iterations. This matrix was a good indicator to evaluate the performance of the RL agent.

Beysolow II (2019: 38) implemented PPO to train an agent that can self-play the first level of World 1 SMB. The author used the general PPO approach by Schulman et al. (2017) and applied a preprocessing by resizing and changing the colour of the images to greyscale. There was no result mentioned by the author since the implementation was for the illustrative purpose; however, the author concluded that the PPO training for SMB was generally will return a good result after 12 hours of training.

C. Summary

Most of the researches related to SMB conducted their implementation in the first level of World 1. Although Beysolow II (2019: 38) has implemented PPO with SMB, there was still no proper research and analysis regarding the performance of the agent in the first level of World 1 using PPO. Nichol et al. (2018) explicitly described the process of training the agent to play in every level of the Sonic The Hedgehog. However, there was no similar works and analysis implemented to different levels of SMB.

III. Solution

The solution to design an RL agent performing better than a normal PPO implementation is presented in this section. The implementation tools and game environment for the systems are also outlined. An overview of the solution is given; subsequently, the detailed solution and the underlying algorithms are described in the sub-sections. Details of testing and validation are also explained the solution.

A. Implementation Tools

Pytorch has been used as the main machine learning library since it is one of the most supported machine libraries, and it can provide maximum flexibility and speed, especially for a small-scale project. Pytorch created the CNN models for actor-critic methods. The process of implementing CNN is straightforward without additional complexity in low-level programming because the implementation is easy and quick due to the functionalities of the Pytorch library. OpenCV, a popular open-source computer vision library, was used to perform some of the preprocessing methods due to the SMB environment, such as resizing the frames and turning the colour of the frames to RGB. Consequently, we used Python language for the implementation since it is the best language for machine learning, and it integrated well with both OpenCV2 and Pytorch. Another python package, numpy, also included performing matrix operation in the implementation.

B. Game Environment

(1) OpenAI Gym

Underlying the SMB environment is OpenAI Gym, a toolkit to create RL environments from various games and developing RL algorithms. The Gym is an open-source library, and the process of integrating the library is easy by exporting the project to our code. The Gym gives access to an RL agent to take specific action in an environment. The environment will return the observation and reward from the action taken. Here are the values returned by the environment:

- Observation- Give the representation of the environment through observation
- Reward- The amount of reward achieved by implementing the previous action
- Done- Return the condition of the state and restart the environment if the result is true.
- Info- Diagnostic information useful for debugging

(2) Super Mario Bros Game

We used the *gym-super-mario-bros* library provided by Kauten (2018) as the environment to train the RL agent. The environment follows most of the rules of the original SMB. The player is only allowed three attempts (lives) to complete the all the eight worlds which contain four levels each. However, we only focused on the first level of World 1 since it is the best level to train an RL agent. The level is easy enough for a human player to play, but through the perspective of an RL agent, the complexity is good to help the agent to gain various knowledge by interacting with the environment.

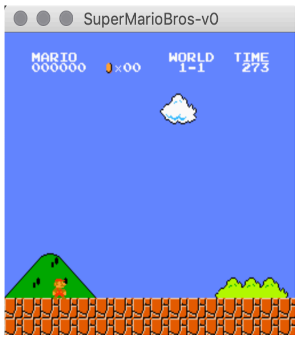


Figure 1: *Screenshot from the SMB environment*

(3) Observation

The SMB environment produces an observation at the beginning of every timestep. This observation is always an image consists of pixels, and every pixel is represented as a numerical value. The images on the screen use RGB value with screen images 320 pixels wide and 224 pixels height. We used the state as the representation of observation in the implementation.

(4) Action

The SMB environment follows the set of actions that can be done through the Nintendo Entertainment System (NES) console's controller. At every timestep, the agent produces an action accordingly to the combination of buttons pressed of the NES console. The SMB environment converts the encoded actions, which are in the form binary to discrete. Therefore, the action space can be described as below:

{{**0**: NONE}, {**1**: RIGHT}, {**2**: RIGHT , A }, {**3**: RIGHT,B}, {**4**: RIGHT, A, B},
{**5**: A}, {**6**: LEFT}}

(5) Reward

During an episode, the environment will return a reward to the agent, and the reward will accumulate until the done function returns true which means the agent has either completed the level or used all the three lives given. The objective of the game is assumed to move as far as right as possible while avoid dying and within a short period of time. The reward function of the environment is defined in Eq. (4):

$$r = v + c + d \tag{4}$$

where r is reward per episode, v is the x-position of the agent, c is the difference in the game clock between frames and d is the death penalty.

The x-position of the agent can be inspected by using the info function from the environment. If the agent moves to the left, it will cause the v value to be negative hence minus the reward gained by the action. For c value, the environment will calculate the difference between the clock time before the step is taken and the clock time after the step is taken. If the agent dies, the agent will be deducted with -15 rewards hence encouraging the agent to avoid death. The reward from the environment is clipped into the range (-15,15). In order to create a suitable reward range for the neural network, the reward is multiplied by 0.05.

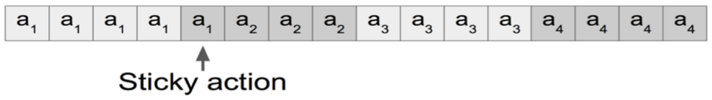
C. Data Preprocessing

(1) Image Preprocessing and Frame Stacking

Working directly with raw SMB frames, which are 320 x 240 pixel images can be computationally demanding, so we applied an image scaling accordingly to the settings introduced by Mnih et al. (2013) to the size of the images. We crop the images to 84 x 84, and we changed the colour of the images to greyscale. The cropping was good enough to capture the playing and provide enough information to the CNN models to read the frames. We also applied frame to the environment since an action taken by the agent is completed usually in 4 frames. By stacking the frame, the CNN models could provide accurate action for the agent.

(2) Stochastic Frame Skip

We used the stochastic frame skipping proposed by Nichol et al. (2018) to introduce a small amount of randomness taken by the agent since SMB's deterministic environment often cause the agent to use a scripted solution. Stochastic frame skipping introduces randomness to the agent's action by applies n actions over 4n frames. We used the probability of 0.4 to delay the previous action by one frame and apply it to the current action. The method used can be described in Figure 2.



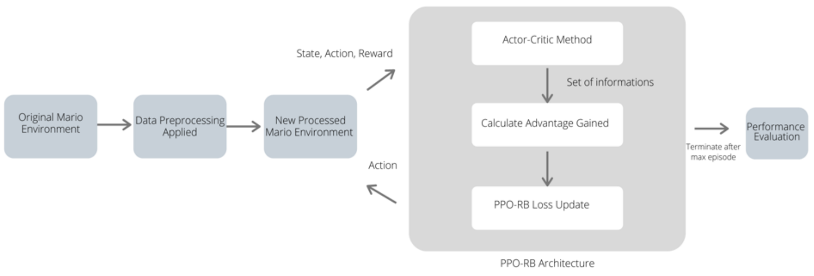
a_1 a_1 a_1 a_1 a_1 a_2 a_2 a_2 a_3 a_3 a_3 a_3 a_4 a_4 a_4 a_4
Sticky action

Figure 2*: Example of an action sequence with a sticky frameskip*

D. Solution Architecture Overview

The overview of the solution architecture is illustrated in Figure 3. The architecture is divided into two parts, the SMB environment settings and the PPO-RB implementation. The first part focused on the settings of the environment, which provides the details regarding the environment of SMB provided OpenAI Gym and *gym-super-mario-bros* from (Kauten, 2018). Data preprocessing method is required in order to filter out unnecessary details and make improvement of the environment. Data preprocessing is important because the data provided from the environment need to be optimal for the agent training using PPO-RB, thus enhancing the performance of the agent and speed up the training time.

The second part focused on the architecture of PPO-RB, which used the same PPO architecture from Schulman et al. (2017) but with an addition from the rollback approach from (Wang et al., 2019). The flow of the process of PPO-RB is briefly shown on the right side of Figure 3. The data that has been processed from the SMB environment is submitted to two Convolutional Neural Network (CNN) models using the actor-critic methods. The setup of the CNN models was created by using Deep Learning library, Pytorch. After the actor-critic methods, the PPO-RB algorithm calculated the advantage gained through the set of information from Actor-Critic methods and update the loss of PPO-RB using the advantage. The process of training the agent was repeated until the maximum number of episodes is achieved. Further details of the architecture are given in the sub-section.



Original Mario Environment
Data Preprocessing Applied
New Processed Mario Environment
State, Action, Reward
Action
Actor-Critic Method
Set of informations
Calculate Advantage Gained
PPO-RB Loss Update
PPO-RB Architecture
Terminate after max episode
Performance Evaluation

Figure 3: *Overview of the solution architecture*

E. PPO With Rollback Operation (PPO-RB) Architecture

(1) Setting of PPO-RB

Nichol et al. (2018) used the hyperparameter settings to the Sonic The Hedgehog game and managed to train an agent that can play the game. Most of the hyperparameters used in the system followed the recommendation by the authors since both SMB and Sonic The Hedgehog has similar characteristics that can influence the progress of agent training such as discrete action space and same end goals. Then we trained eight workers in eight different environments running in parallel to speed up the process of training the agent. The higher the number of workers, the better the performance, but, however, due to the limitations in computational resource, eight workers were the highest for the implementation. A mini-batch update was applied to the implementation which divided the datasets into small batches used to measure the model error and update the model coefficient. Keskar et al. (2017) mentioned mini-batch with larger size will cause a degradation in the quality of the model, and smaller size leads to poorer generalization. Hence, we used 64 mini-batches that is the middle ground between 32 and 128.

(2) Actor-Critic Methods

As mentioned in the overview subsection, PPO-RB has several layers of processes in the system. The first layer of the architecture consists of two CNN models which are actor-critic methods. The actor-critic methods are the combination of both policy-based and value-based approach to train an RL agent (Surton and Barto, 1998: 257). By combining both approaches, the architecture gains the benefit of achieving a faster convergence in stochastic environments while value-based is more sample efficient and steady. The methods are also mainly to overcome both of the drawbacks by complementing each other.

The actor model acts as a function approximator, and its role is to generate the best action for a given state. We applied a CNN model to create the actor model with input of dimension (8, 84, 84, 4). The first element of the input is the number of workers (8), second and third elements are the height (84) and width (84), and the last element is the number of frames stacked (4). The actor model updates the model parameters θ in the direction, as suggested for the value by the critic model.

The critic model is another function approximator, which evaluates if the action taken by the actor leads to the environment is improved whilst giving feedback to the actor. The CNN model takes the state and the action by the actor by concatenating them. It outputs a Q-value of the action taken in the previous action. Both the actor and critic models perform gradient ascent to update both their weights. This causes the actor to learn in taking the best action, and the critic will be getting better in evaluating the action as time passes. Both the models use TD Learning by updating the weights at each step rather than updating the weights at each episode. Figure 4 shows the brief architecture of actor-critic methods.

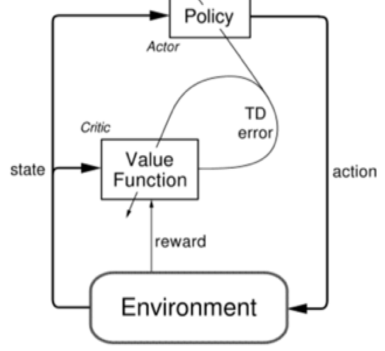


Figure 4: *Actor-Critic Architecture (Surton and Barto, 1998: 258)*

Fortunato et al. (2019) introduced NoisyNet by adding parametric noise to its weight. By inducing stochasticity into the agent's policy, efficient exploration can be aided. We implemented NoisyNet to the actor-critic CNN models in order to generate stochasticity to our implementation.

(3) Calculate Advantage Using Generalized Advantage Estimation (GAE)

Before the transition from the first layer to the second layer, the algorithm keeps training the agent and observes the reward gained within 256 number of PPO steps. After the number has been reached, the algorithm stores arrays of information:

- Mask value- An array of Boolean variables containing the opposite of Done variable. The opposite Boolean is taken because the algorithm only wants the state that is still in-game.
- Rewards- An array of rewards gained from the training steps.
- Values- An array of value functions returned from the Critic model
- Actions- An array of actions returned from the previous actions in training
- Log Probabilities- An array of log probabilities gained from the Actor model.

These arrays are then passed to the second layer to update the value of advantage gained from the previous PPO training steps.

Advantage can be described as a way of calculating how much better we can be by taking a specific action when the agent in a given state. The goal is to use the received bonus at every phase of the time and measure how much benefit that able to achieve from taking the previous action. If for example, the agent managed to jump over an obstacle, quantifying need to be done to evaluate how much better off the agent has been for taking that action, not just in the short term but also over a longer-term. This way, even if the agent failed in jumping over the obstacle in the next time phase, the agent still looks into the longer future to see if it is able to be successful in jumping over after that action. GAE was used in our implementation since it is the state-of-the-art implementation in calculating the advantage (Schulman et al., 2015). By implementing GAE, our implementation achieved significantly faster policy improvement compared to the pure actor-critic methods.

(4) PPO With Rollback function

Based on Figure 3, the last layer takes the calculated advantages from the second layer and updates its loss function alongside with the information from the first layer using PPO. Wang et al. (2019) mentioned that PPO could not strictly confine the likelihood ratio within the clipping range. The likelihood could be extended beyond the clipping range $[1-\epsilon, 1+\epsilon]$ which is not desirable in stabilizing the policy. They addressed the issue by introducing rollback operation once the likelihood ratio exceeded the clipping range. The authors demonstrated PPO-RB achieved higher reward in compared to general PPO in performing Mujoco tasks. Our implementation with rollback operation managed to achieve a better performance than usual PPO algorithm. We demonstrated the result in our result section.

F. Testing and Evaluation

When action from the model is submitted to the environment, the rewards and distance reached can be extracted. This information is useful to evaluate the performance of the algorithm. The first evaluation is to check that the solution managed to complete the first level of World 1 within 30 hours of training time. The time taken to finish training was recorded in order to check the speed of the algorithm in training the agent. The shorter the time taken indicates that the algorithm performs well. The second evaluation was using cumulative reward per iterations of training. The graph with better slope indicates how good the policy is after the algorithm has stabilized. The last evaluation is by testing the performance. We used the models that have been trained by the three algorithms and performed testing by evaluating the maximum distance and mean distance achieved by the agent. Higher maximum and mean distance resulting in the algorithm is able to train the agent quicker and producing a high-quality agent.

IV. RESULTS

In the section, the results of the training and the comparison between our implementation and other implementations are presented. Experiment settings and performance analysis are explained in this section. The parameter configurations are shown in Table 1:

Hyper-parameter	Value
PPO Epochs	10
PPO Steps	256
Discount (γ)	0.5
GAE parameter (λ)	0.95
Clipping parameter (ε)	0.2
Entropy coefficient	0.001
Rollback parameter	0.5

Table 1: *List of Hyperparameters*

We conducted the training and experiments thrice in order to have accurate results and to eliminate arbitrary error results. The mean of the results is calculated and listed in the table. The system configuration for the evaluation methods used the Google Colab platform with a single core hyperthreaded Xeon Processors @2.0Ghz CPU, Tesla T4 GPU, 12 GB RAM. For evaluation, we conducted the experiment with three different algorithms, 1) PPO with preprocessing and rollback function, 2) PPO with preprocessing and without rollback, and 3) general PPO without any preprocessing and rollback function.

A. Agent Level Completion

Due to the limitation of 12-hours per session from Google Colab, we conducted the training in many sessions in order to train the agent. However, we set a time limit of 30 hours to conclude that the implementation unable to train the agent. Therefore, the goals of the RL algorithms is to train an agent that can reach the end of the level within 30 hours. In order to identify whether the agent managed to complete the level, the agent requires to reach the maximum distance of 3175. The PPO-RB With Preprocessing agent managed to complete the training in 20 hours while the PPO With Preprocessing managed to complete the agent in 24 hours. However, for the normal PPO, the agent unable to complete since it stuck at a local optimum. The results of the training for PPO-RB With Preprocessing, PPO With Preprocessing and PPO is described in Table 2.

Method	Agent Completion Within 30 hours	Time taken to complete (hours)
PPO-RB With Preprocessing	Success	20
PPO With Preprocessing	Success	24
Normal PPO	Fail	-

***Table 2**: Agent Level Completion Result*

B. Agent Cumulative Reward

We set the 12-hours mark as an indicator to evaluate the performance of the agent during training. 12-hours is chosen since most of the agents converge (stuck at a local optimum) at the 12-hours mark. We calculated how many rewards gained by each agent per iteration. The result in Figure 6 showed that both PPO-RB With Preprocessing agent and PPO With Preprocessing agent managed to have a steady increase. However, PPO agent failed to have a steady increase indicating that the agent constantly receiving negative reward hence unable to learn well.

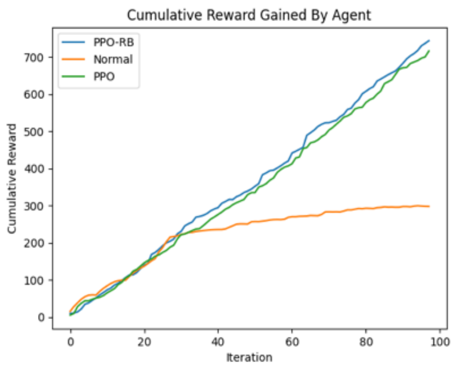


***Figure 5**: Agent Evaluation Using Cumulative Reward Gained Per Iteration*

C. Testing Performance

We used the models trained at the 12-hours mark, around 100 iterations to identify the performance in terms of quality and speed. Mean distance and maximum distance gained per iterations are used to evaluate the performance. According to the result, our implementation, which is PPO-RB With Pre-Processing performed 10 % better compared to PPO With Preprocessing and performed 50% better than normal PPO. The result of our testing is described in Figure 5.

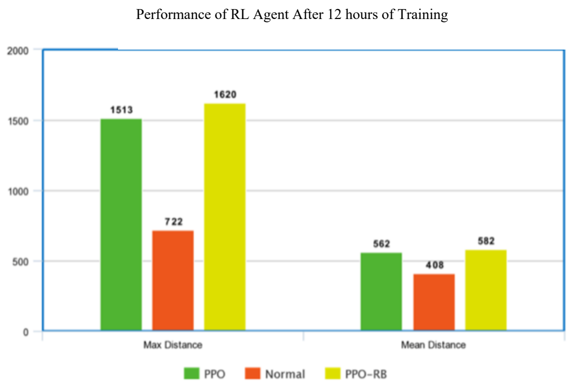


***Figure 6:** Agent Evaluation Using Max Distance and Mean Distance*

V. EVALUATION

A. Solution Strengths

We have demonstrated that we have successfully trained an agent that can complete the first level of SMB World 1 using our implementation. Both our proposed solutions; PPO with Preprocessing and PPO-RB with Preprocessing able to train an agent that able to solve the level while the normal PPO is struggling to complete the level. Furthermore, our final solution, which is PPO-RB with Preprocessing managed to train the fastest, demonstrating the capability of our model, creating a quality agent. The pillar of the problems, Markov Decision Process, behind RL has been tackled by our agent since we managed to train an agent that can make a decision based on the given state. Even though the environment of SMB is large and complex, our agent managed to interact and learn to complete the level. The solution provided by our agent is also good since it managed to gain more reward meaning able to learn as time passed.

We also demonstrated that by implementing preprocessing to the state-of-the-art PPO implementation, we managed to increase the performance by 40%. We managed to improve the PPO implementation further by introducing rollback to PPO with Preprocessing. The method improved the performance by 10% compared to PPO with Preprocessing hence indicating we have successfully implemented a system that performed better than the normal implementation. Our main goal, which is to improve the implementation of PPO is successful since the result produced to improve the performance of the normal implementation.
We can conclude that the performance of our implementation performed better due to:

- The introduction of randomness in the frames and the neural network during preprocessing. This helped the agent to take any action to overcome a local optimum.
- By scaling the size of frames and making reduce to the colour to greyscale, we were able to train the agent within the time 30 hours limit.
- By introducing a rollback operation, we managed to improve the performance of PPO by 10% due to stability introduced by the rollback operation.

B. Solution Limitations

Although we have successfully trained an agent to play the first level of World 1 using our implementation, the project was unable to progress beyond the minimum objective. One of the reasons was that we spent most of the times by trying to implement the solution to level 1 World 1. The learning curve to learn about RL was steep since the approach to solve RL problems were different from the normal Machine Learning techniques. Apart from that, the process of training a good agent was time-consuming since the agent needs to interact with the environment with zero knowledge. Furthermore, the complexity of the training also increased due to the dimension of the SMB environment, which was huge and challenging for the agent. Hence, in order to evaluate the agent performance, at least 12 hours were allocated for each training session.

Unfortunately, when the project was about to progress to an intermediate objective, the project halted due to unavoidable circumstances. The access to computational resources from the university was unavailable due to the closing of the university. The usage of Google Colab helped the progress of our project compared to using a regular machine without any graphic card. However, the session limit of 12-hours halted the progress of the training since we needed to wait for hours to use the service back. The availability of the service also depends on the usage of different consumers.

To implement the intermediate objective, an agent must be assigned each to 32 levels of SMB. The policy that we trained in the first level World 1 could not be used for different levels since the generalization of the policy was bad since the environments of each level SMB was different. The hyperparameters must also be adjusted accordingly to each level.

In order to improve the generalization of our agent, we proposed the meta-learner implementation as our advanced objective. However, we were unable to reach this phase since we could only spend time on the minimum objective. Another thing that needed to be considered based on the implementation by Nichol et al. (2018), they used 189 parallel workers which required powerful computational resources. Assuming we reached the advanced objective, lacking times and resources might cause the objective still could not be achieved.

C. Overcoming Limitations

To demonstrate a good outcome for the project, we demonstrated a thorough analysis of our PPO implementation. We conducted various evaluations to make sure we produced the best result for the project. We also repeated our evaluations thrice to eliminate any error and to gain accurate results.

In order to make sure we trained the agent completely without being affected by the time limit from Google Colab, we made sure our model weights were saved, and the progress of the frames and the number of iterations were also being monitored. We wanted to resume the training progress without losing any vital data that might hinder our training progress. The approach worked well, especially to overcome the time limit. The approach also solved the problem of losing the progress data when Google Colab disconnected during the agent's training.

In the early stages of this project, we spent most of the time understanding the concept of RL and exploring the best framework to implement in the project. In order to acquire a greater understanding regarding RL, we used online forums such as *discuss.pytorch.org* to reach out to people that have a broad knowledge of RL. We were able to discuss and solved our problems regarding the project using this method.

VI. CONCLUSION

We have successfully achieved our minimum objectives by successfully trained an RL agent to play the SMB level 1 World 1 by using the improved version of the PPO algorithm, which was PPO-RB. We also introduced data preprocessing to improve the performance of the agent. The PPO-RB approached overcame the problem of instability of the PPO in updating the policy hence producing a better result than the regular implementation. Our result proved that our method managed to improve the state-of-the-art PPO algorithm in SMB. We managed to train the agent in 20 hours with our solution, faster than the PPO with Preprocessing. We also increased the performance of the agent drastically compare to PPO without any preprocessing by achieving further maximum and mean distance. Since there is no prior work that provides an analysis of performance for PPO agent in SMB, we demonstrated the result in our work.

Our project can be extended in many ways. Our intermediate objective can be implemented to train agents that can self-play for all the 32 levels. Furthermore, we can shorten the time taken in training the agent for each level by introducing a meta-learner. Meta-learner will improve the generalization of PPO. In this project, the goal of the agent is to complete the level by keeping moving to the right and avoiding death. In the future, an introduction of extra reward function such as collecting coins and killing enemies is useful to create an RL agent that can reach the performance of a human player. Apart from that, we would like to train the agent with powerful computational resources so that we can provide the best and optimum result.

Extending the project by comparing our approaches with different state-of-the-art algorithms are also beneficial to evaluate the performance of our implementation.

REFERENCES

Bellemare, M.G., Naddaf, Y., Veness, J. and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, [online] 47, pp.253–279. Available at: https://arxiv.org/pdf/1207.4708.pdf [Accessed 2 Aug. 2019].

Beysolow II, T. (2019). *Applied Reinforcement Learning With Python*. Berkeley, CA: Apress.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016). *OpenAI Gym*. [online] arXiv.org. Available at: https://arxiv.org/abs/1606.01540.

Choi, J., Laibson, D., Madrian, B. and Metrick, A. (2009). Reinforcement Learning and Savings Behavior. The Journal of finance. 64. 2515-2534. 10.1111/j.1540-6261.2009.01509.x.

Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C. and Legg, S. (2019). Noisy Networks for Exploration. *arXiv:1706.10295 [cs, stat].* [online] Available at: https://arxiv.org/abs/1706.10295.

Grand, G. and Loughlin, K. (2018). *Reinforcement Learning in Super Mario Bros. CS 182 Final Project Report*. [online] Available at: http://www.gabegrand.com/files/super_mario_rl.pdf [Accessed 10 March 2020].

Gottesman, O., Johansson, F., Komorowski, M. *et al*. Guidelines for reinforcement learning in healthcare. *Nat Med* 25**,** 16–18 (2019). https://doi.org/10.1038/s41591-018-0310-5

Kauten, C. (2018). *Kautenja/gym-super-mario-bros*. [online] GitHub. Available at: https://github.com/Kautenja/gym-super-mario-bros [Accessed 1 January 2020].

Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P.T.P. (2017). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv:1609.04836 [cs, math].* [online] Available at: https://arxiv.org/abs/1609.04836.

Liao, Y., Yi, K. and Yang, Z. (2012). *CS229 Final Report Reinforcement Learning to Play Mario*. [online] Available at: http://cs229.stanford.edu/proj2012/LiaoYiYang-RLtoPlayMario.pdf.

Mnih, V., Badia, Adrià Puigdomènech, Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D. and Kavukcuoglu, K. (2016). *Asynchronous Methods for Deep Reinforcement Learning*. [online] arXiv.org. Available at: https://arxiv.org/abs/1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. [online] arXiv.org. Available at: https://arxiv.org/abs/1312.5602.

Nichol, A., Pfau, V., Hesse, C., Klimov, O. and Schulman, J. (2018). Gotta Learn Fast: A New Benchmark for Generalization in RL. *arXiv:1804.03720 [cs, stat].* [online] Available at: https://arxiv.org/abs/1804.03720.

Openai, Berner, C., Brockman, G., Chan, B., Cheung, V., Psyho, quot;, Dębiak, quot;, Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pondé, H., Pinto, O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F. and Zhang, S. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. [online] Available at: https://cdn.openai.com/dota-2.pdf [Accessed 13 Mar. 2020].

Schulman, J., Levine, S., Moritz, P., Jordan, M.I. and Abbeel, P. (2015). *Trust Region Policy Optimization*. [online] arXiv.org. Available at: https://arxiv.org/abs/1502.05477.

Schulman, J., Moritz, P., Levine, S., Jordan, M.I., & Abbeel, P. (2015). High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR, abs/1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. [online] arXiv.org. Available at: https://arxiv.org/abs/1707.06347.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), pp.484–489.

Sutton, R.S. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge, Ma; London: The Mit Press.

Wang, Y., He, H., Wen, C. and Tan, X. (2019). *Truly Proximal Policy Optimization*. [online] Available at: https//arxiv.org/pdf/1903.07940 [Accessed 10 March 2020].

Wu, Y. and Tian, Y. (2017). *Published as a conference paper at ICLR 2017 Training Agent for First-Person Shooter Game with Actor-Critic Curriculum Learning* [online] Available at: https://pdfs.semanticscholar.org/add7/b8b65355d5408a1ffb93a94b0ae688806bc4.pdf?_ga=2.260406477.1198601304.1589141964-1001151123.1588820767 [Accessed 10 May 2020].