Super Reinforcement Bros: Playing Super Mario Bros with Reinforcement Learning

Nan ZHANG
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
`zn020@ie.cuhk.edu.hk`

Zixing SONG
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
`zxsong@cse.cuhk.edu.hk`

Abstract

We plan to apply and adjust some well-known reinforcement learning (RL) algorithms to train an automatic agent to play the 1985 Nintendo game Super Mario Bros under a speedrun rule. The agent may learn several control policies from raw pixel data by using deep reinforcement learning. By the end of the project, we expect the model to perform comparably to or top human players in a given stage. Our code repository is available at `https://github.com/AnthonySong98/Super-Mario-Bros-PPO`.

1 Problem Definition

Since 1981, in the gaming industry, the red-hatted Mario plumber has monopolized the industry to protect the princess from excessively smart turtles or apes while collecting oversized coins along the way. Even though, there is still some room for competition in such an excessively dominated market, hence the project is intended to challenge the salvific plumber industry by incorporating reinforcement learning to see if Mario's work can become totally automated. More specifically, we plan to use some popular reinforcement learning algorithms, such as ϵ-greedy Q-learning, SARSA and etc., to train our agent to clear all the stages within the shortest time in the game of Super Mario Bros.

2 Related Works and Challenges

2.1 Deep Q-Learning

Deep reinforcement learning has been performing decently in games in recent works. In [7] a deep Q-learning network (DQN) with experience replay is constructed to play the Atari game and it can achieve scores comparable to human plays. In the following years, advancements in reinforcement learning introduce techniques such as double DQN and dueling DQN. While they boost the performance of reinforcement learning agents by achieving higher rewards, they also cast light on implementations in games. Later [4] trained a DQN with SiLU and dSiLU as activation functions in the place of ReLU or sigmoid functions and benchmark the network with the Tetris games. Results show that in a stochastic SZ-Tetris game, SiLUs significantly outperformed ReLUs, and that dSiLUs significantly outperformed sigmoid units. These games have the same objective to get as many points as possible, which can be interpreted as to stay alive as long as possible. While the introduction of deep neural networks improves the performance of agents in these games, we would like to observe the progress in a setting where the agent is expected to finish a task as fast as possible.

2.2 Continuous control algorithms

Since deep Q-learning may produce a learning curve with great oscillations, it is worthwhile to see how policy gradient methods perform, which are more stable in training. PPO [9] and A3C [6] are algorithms previously validated in environments with continuous action spaces. According to the results provided in the original papers, they generally perform better than a DQN, even in an environment with discrete action space. The action space in Super Mario Bros can be considered as continuous since longer pressing time on the jump button results in a higher position. Based on this assumption, we can testify the capability of PPO and A3C in the environment.

2.3 Challenges

As part of this plan, we originally intended to use classical Q-learning, since much game knowledge is available for handcrafted applications. The effectiveness of this model, however, depends heavily on the quality of the representation of the extracted features. It will certainly create an exceedingly large state and action space if all possible permutations of Mario's world are directly represented. For better performance, we also experimented with continuous control algorithms, namely PPO and A3C, which are more capable to solve the problems in the Mario's world. The difficulty in action and state representation is reduced by the continuous control algorithms and the key point here is to select a suitable probability distribution for the data. Another noteworthy challenge in reinforcement learning stems from the fact that rewards are sparse and time-delayed. When the agent earns a reward, we must determine which of the preceding actions played a role in getting that reward, and to what extent [7]. Despite these challenges, a convolutional neural network updated with stochastic gradient ascend can reasonably well be compared to a learning human player.

3 Environment and Data

The general goal of the game is to control Mario to reach the finish line, and gain a high score on the way by collecting items and beating enemies. However, in this project, we only focus on how to finish the stage as fast as possible without considering how to get the highest score. The gym environment is adopted from [5] with our modifications. We will explain the structure of the environment in the following subsections

Notes on Warp There is a hidden feature in Super Mario Bros called "warp". It's a teleportation zone that allows a player who reaches here to skip certain stages [1]. If an agent can utilize warps, it can tremendously reduce the time to beat the whole game by skipping some stages. The enhancement enables an agent to compete with human players under the "Any%" rule or even discover a more efficient combination of warps to beat the game.

However, in our training environment, we ignore warp feature to simplify the problem. So only the warpless mode in this game is investigated.

3.1 Action: Joystick and Buttons

In the original game, every action that Mario can take is executed by solely or combinedly turning the joystick and pressing the "A" and "B" buttons on a controller. In the emulated environment, the combinations are associated with keywords described as four directions, "A", "B" and "NOOP" when nothing is triggered. In a speedrun setting, staying at a spot is the least useful action as the time is ticking. Therefore, we remove the actions that are not ideal to the project purpose, which leaves the remaining available actions to be:

- **Noop**: do nothing
- **A**: jump
- **Down**: duck, enter a pipe or climb downwards on a beanstalk
- **Up**: climb upwards on a beanstalk
- **Right**: move rightwards
- **Right + A**: jump or swim upwards while moving rightwards

- **Right + B**: dash or throw fireballs (only when Mario is in "Fireball" status) while moving rightwards
- **Right + A + B**: combination of the above two

There are also the corresponding leftwards actions to the last four actions. In total, 12 actions are available.

3.2 Agent: Mario

In the game, Mario has three status: "small", "tall", and "fireball". He can upgrade to "tall" by eating a mushroom and to "fireball" by eating a flower. Yet, he will lose the current status and return to "small" if attacked by enemies. When Mario is "tall", he has twice the size of a "small" Mario. When he is "fireball", he can shoot at most two fireballs at a time which kill enemies.

3.3 Environment: Stage

As a platform game, the stage is the foundation where the controllable character runs. In this project, we consider there are 4 types of platform assets and 4 types of items. platforms (hill, pit, cannon, and pipe) and items (coin, mushroom, bricks, flower). The platform assets contain:

- **Ground**: the base level of a stage
- **Pit**: the hole in the ground
- **Cannon**: a roadblock that shoots a straightly flying enemy called "Bullet Bill" once a while
- **Pipe**: a channel that teleports Mario to a hidden stage

For most of the time, Mario runs on the ground and jumps to avoid the pits. If he falls into a pit, he dies. Cannon works almost the same as a normal obstacle which Mario can touch on stand on, unless it produces enemies periodically. Pipe is a very important platform asset because Mario can transport to another spot by going through pipes. By doing this, a player can instantly pop up at the spot close to the final destination skipping all the grounds and gaps in between the two pipes, which greatly reduces the time.

The items contain:

- **Coin**: gain points
- **Mushroom**: change Mario from "small" to "tall" and gain points
- **Flower**: change Mario from "small" or "tall" to "fireball" and gain points
- **Brick**: generate the above 3 items or break when hit

Unlike the platform assets, items can disappear from the stage by certain player actions. For a coin, a mushroom and a flower, they disappear when Mario consumes them. A brick, regardless of its type, can also break and disappear. However, not all bricks are breakable and the breakable bricks look exactly the same as unbreakable ones until they are hit. If Mario consumes a mushroom, he becomes "tall" from "small". If he consumes a flower, he becomes "fireball" from either "tall" or "small". He will neither change status again nor store the item for a chance to get hit if he eats the same item. He will not revert to "tall" in "fireball" status even if he eats a mushroom, either.

4 Proposed approach

In this section, some key elements in RL under the setting of speedrun in Super Mario Bros. are covered in detail with definitions and simplifications so that some potential RL algorithms can be easily applied.

4.1 State: Mario, Action and Environment

As the agent is moving to different positions in a stage, transitions will be generated by its interactions with the environment. The transitions, which can be detected by our agent, constitute the state of the environment. The new states observed by our agent can produce a signal of "reward". Combining the actions taken by the agent, the transitions between states and the possible rewards for the transition, the agent starts laying the foundations for a RL model. We assume that our agent does not have all the information of the environment (e.g. the exact remaining distance to the destination flagpole), and it only possesses the knowledge shown on the screen at the current state (i.e. frame). Key transitions available in a state are:

- **Position**: the coordinates of the agent
- **Time**: the remaining time on the clock
- **Action**: the stance of the agent
- **Status**: the living status and the functional status ("small", "tall" and "fireball") of the agent
- **Flagpole**: the status of the destination flagpole (active if reached)

Among them, the position, time, action information is used to derive the next state and to calculate the "reward" of a given state, whose details are provided in the next subsection. The status information and the flagpole information is used to determine if the game is over.

4.2 Reward: Shortest Time

We assume the objective of the game is to beat the game as fast as possible [2]. Therefore, the most important things in the metrics are the horizontal distance Δx from the current spot to the destination flagpole and the time difference Δt between starting time and ending time. Besides the two, we also consider whether Mario eventually reaches the flagpole f matters. We want to encourage trajectories that actually beat the game, but it should not be influencing the overall reward too much because some trajectories that does not make it to the destination may be a better option if some missteps are eliminated. The same logic applies to a death penalty d, which we use to alert the agent to avoid that spot in the next episode.

Distance: Let x_0 be the horizontal position of the agent before the step, and x_1 be the horizontal position after the step.

$$\Delta x_n = x_{n+1} - x_n \tag{1}$$

If the agent moves rightwards $\Delta x > 0$. If the agent moves leftwards $\Delta x < 0$. If it stays at the same position $Deltax = 0$.

Time: Let t_0 be the time on the clock before the step, and t_1 be the time on the clock after the step. The time starts at 300 seconds for a newly started stage and decreases every second.

$$\Delta t_n = t_n - t_{n+1} \tag{2}$$

Since t_0 is always greater than t_1, Δt is always a negative value.

Flagpole: Let ρ be a positive constant.

$$f_n = \begin{cases} \rho & \text{if the agent is getting the flagpole} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Death: Let σ be a negative constant

$$d_n = \begin{cases} \sigma & \text{if the agent is dying} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Total reward: Let r_n be the total reward at the n-th step and it adds (1) to (4) together.

$$r_n = \Delta x_n + \Delta t_n + f_n + d_n \tag{5}$$

5 Experiments

To start with, a two-phase plan based on the capacity of the agent is as follows.

5.1 Train on an individual stage

We use the very first stage, world 01 - stage 01, as our primary experiment stage. The agent will play the stage for 25,000,000 frames in total, trying to maximize the reward for an episode. For the action space, we enabled the rightward movements only to simplify the training process. Because glitches or warp zones are not allowed under our rule set, an agent only needs to move right and to determine the right timings for a jump in a satisfactory run.

We trained PPO agents with the following hyperparameters as default:

- Action type: simple;
- Learning rate η: 1e-4;
- Discount factor for rewards γ: 0.9;
- GAE Parameter τ: 1.0;
- Entropy coefficient β: 0.01;
- Clip parameter for Clipped Surrogate Objective ϵ: 0.2

5.1.1 Clip parameter ϵ

A smaller clip parameter slowed down the progress to convergence by tens of episodes, but the stability was promoted and it demonstrated a more promising trend to higher rewards judging from the last few episodes.

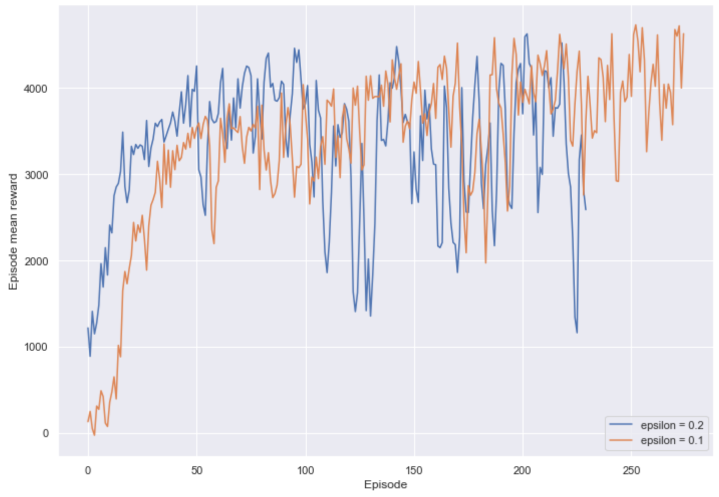


Figure 1: Changes of mean reward of PPO agents with different clip parameter ϵ during training in world 1 stage 1

5.1.2 Discount factor for rewards γ

With a larger discount factor, an agent performed much worse than the one with default hyperparameters. The reason is probably that the initial episodes returning unsatisfactory rewards (i.e. the agent died early or got stuck) had stronger influence on the learning process.

5.1.3 Action type

The actions related to moving left were added to the action space in this experiment. Although it achieved similar rewards, there were more oscillations from time to time, because the agent might be

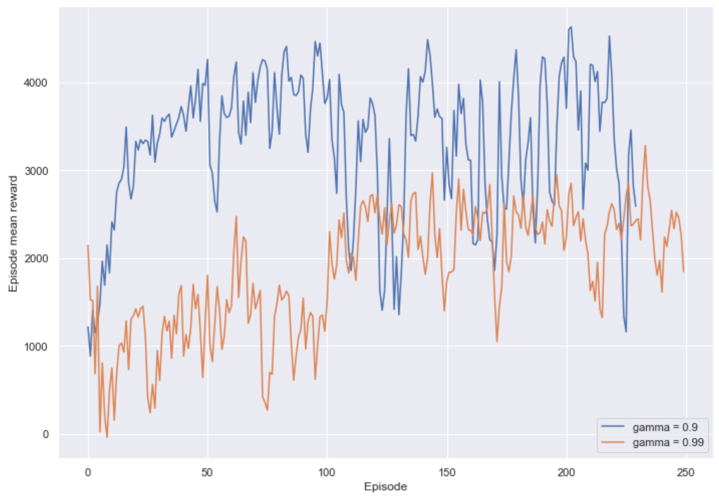


Figure 2: Changes of mean reward of PPO agents with different discount factor γ during training in world 1 stage 1

confused about the moving direction when it reached a spot whose structure does not resemble the one of previous parts and it needed more explorations to find the optimal solution.

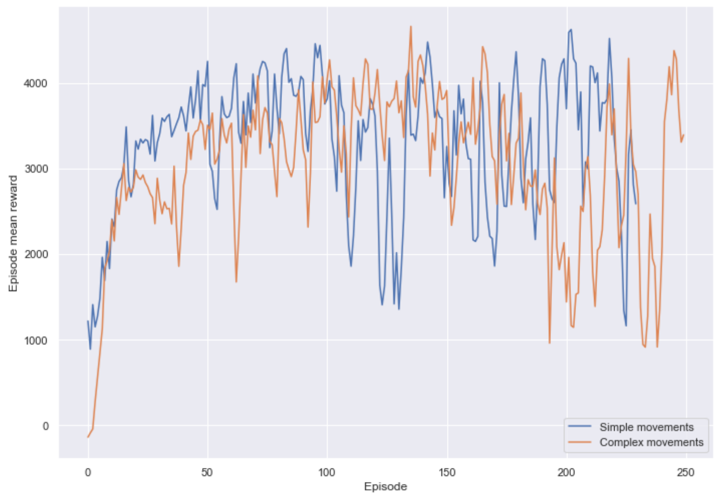


Figure 3: Changes of mean reward of PPO agents with different action type during training in world 1 stage 1

5.2 Comparison with human players

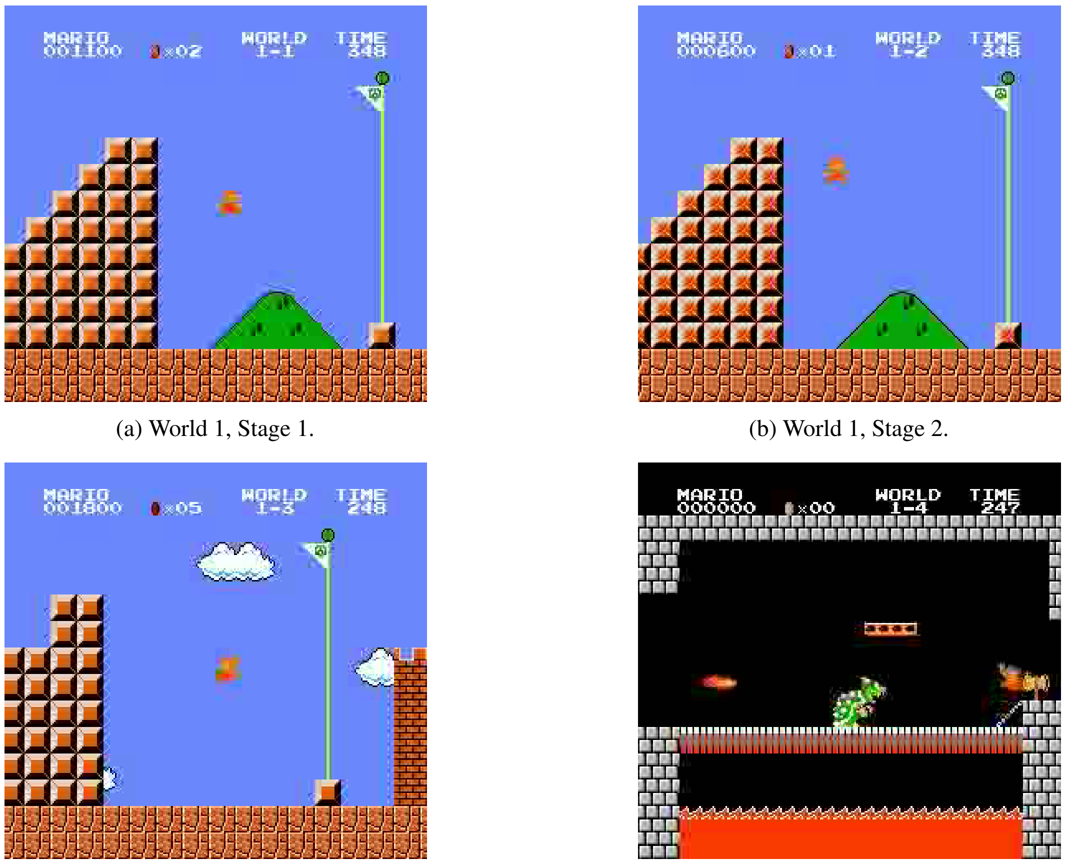
We use PPO to train the agent and successfully complete all four levels in World 1. Figure 4 shows the exact moment when the trained agent complete each level.

Table 1: Comparison with human players in warpless mode

Level	Trained Agent	World Record	Average Human Players
World 1, Stage 1	52	52	64
World 1, Stage 2	53	50	63
World 1, Stage 3	52	42	58
World 1, Stage 4	54	40	57

Videos of Trained Agent's Performance Note that videos are generated during the test phase as well. However, due to the limitation of the paper, we attach the following link for detailed review. Only members affiliated with CUHK can have access to it.

`https://mycuhk-my.sharepoint.com/:f:/g/personal/1155154521_link_cuhk_edu_hk/EkfahCbaTCJBoe_gWfQiqccBP-aXrltI5_ezqkfsKZpChg?e=z8Augz`


MARIO 001100 ×02 WORLD 1-1 TIME 348
MARIO 000600 ×01 WORLD 1-2 TIME 348
MARIO 001800 ×05 WORLD 1-3 TIME 248
MARIO 000000 ×00 WORLD 1-4 TIME 247

(a) World 1, Stage 1.

(b) World 1, Stage 2.

(c) World 1, Stage 3.

(d) World 1, Stage 4.

Figure 4: The trained agent successfully completes first four levels in Super Mario Bros 1985 NES.

In order to further illustrate how well our trained agent performs in these solved levels in the game, we compare the timing performance with the world record in warpless mode where the shortcut tunnel is also not allowed. The total time to complete one level is defined as the difference of the end time and the start time in the right top corner of the video frame. The results are summarized as Table 1.

We can see from Table 1 that our trained agent performs much better than average human players. Even in some levels, the agent even show a comparable performance result with the current world

Table 2: Hyper-parameters Setting For 4 Solved Levels

Level	Learning rate η	Discount factor γ	Entropy coefficient β	Clip parameter ϵ
World 1 Stage 1	1e-5	0.9	0.01	0.2
World 1 Stage 2	1e-5	0.9	0.01	0.2
World 1 Stage 3	2e-4	0.9	0.01	0.2
World 1 Stage 4	1e-5	0.9	0.01	0.2

record. To make the comparison fair, we only compare the performance of the trained agent with that of human player world record under the warpless mode[1].

Also, the hyper-parameter setting for the above four agents are listed in Table 2. We can see from Table 2 that we can easily train the agent to complete the levels by only changing the learning rate. Other parameters can simply be set to the default value.

5.3 Trained with other RL models

5.3.1 Failure of DQN

Currently, only DQN-based agent is trained and tested within the same stage so far. It is worth noting that the environment only sends reward-able game-play frames to agents; No cut-scenes, loading screens, etc. are sent from the NES emulator to an agent nor can an agent perform actions during these instances to further simplify the problem. The changes of reward and loss are illustrated in Figure 5. We can see from Figure 5 that the training process is not stable which corresponds to the properties of DQN model and the complex environment of the game.

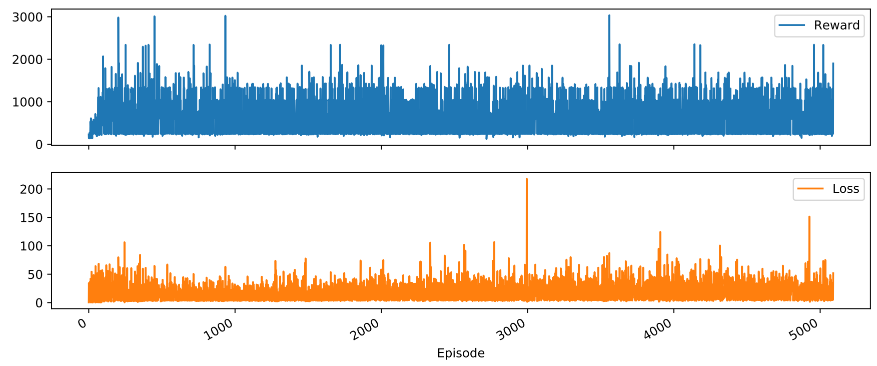


Figure 5: Changes of Reward and Loss of DQN-based agent during training in stage 1

After the training process, we test the actual performance based on the learned policy from the DQN model. In this experiment, the agent is still tested in the same stage of the environment for repeated 100 times. We summarize the results by constructing a histogram of the obtained scores by the trained agent to show how far the agent has actually gone through in the environment. The final result is shown as Figure 6. It is easy to see from Figure 6 that even though the majority of agents only travel not far from the starting point, a few of agents successfully go through quite far enough by avoiding the obstacles and beating the enemies along the way.

Recorded Videos Furthermore, we monitor the whole testing process and record some footage on the successful and failed moments of the agent playing the game. The shared link is provide [2]. Please

[1] `https://www.speedrun.com/smb1#Warpless`

[2] `https://mycuhk-my.sharepoint.com/:f:/g/personal/1155154521_link_cuhk_edu_hk/Euee6DmLPL9JhwK4XhMg6VIBgxyb5U41HhpNQlLaSD6bSw?e=fIHx73`

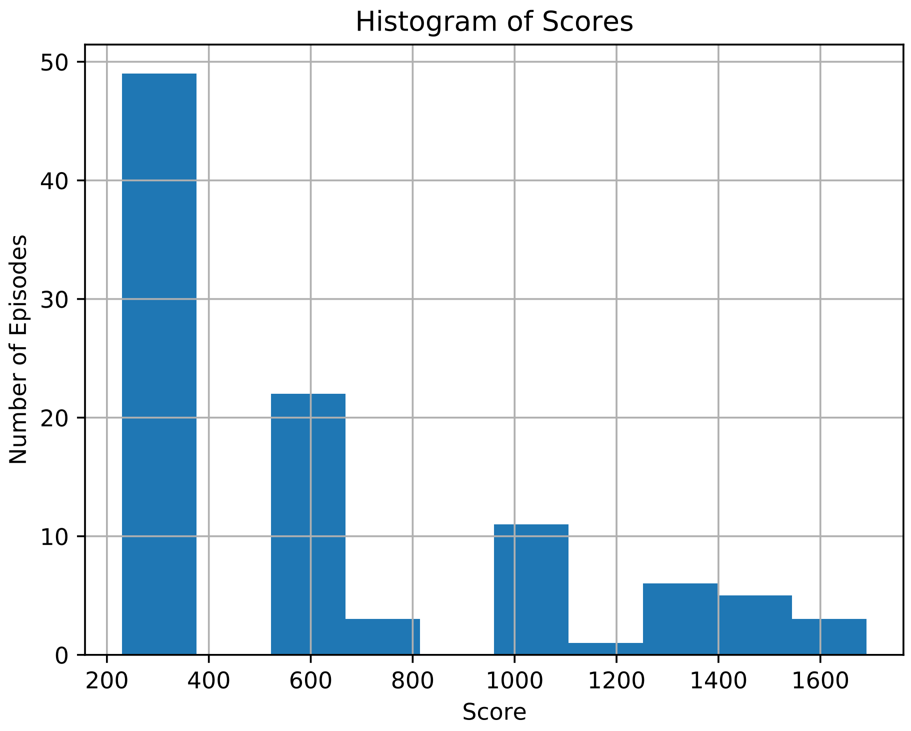


Figure 6: Changes of Reward and Loss of DQN-based agent during training in stage 1

note that only the members from The Chinese University of Hong Kong (CUHK) can have access to it.

6 Conclusions

We have successfully trained PPO agents to beat the whole World 1 of Super Mario Bros. Their performance is comparable to top human experts. The influence of selected hyperparameters and a quick setting for a robust agent are discovered. However, DQN-based agents cannot fulfill the task as expected, which even failed to finished half of World 1 Stage 1. Lastly, some recorded videos to monitor the trained agents are presented to further show the excellent performance of them.

7 Future Development

Super Mario Maker 2 is a game system where a player can craft stages using the assets of Super Mario Bros and let others challenge them [3]. It is worth training our agents with large player-made stage sets to observe their performance on unseen extremely hard stages. A mature agent is expected to benchmark human performance in certain stages (e.g. a stage where there is only one way to reach the goal), just as what [8] did in the Dota 2 game.

References

[1] `https://www.mariowiki.com/Warp_Zone`, 2020. Accessed: 2020-09-25.
[2] `https://www.speedrun.com/smb1`, 2020. Accessed: 2020-09-25.
[3] `https://www.mariowiki.com/Super_Mario_Maker_2`, 2020. Accessed: 2020-09-25.
[4] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.
[5] C. Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.