"Super Mario Bros ML model using Reinforcement Learning"

Musa Zaheer

Department of Computer Engineering, Information Technology University, Lahore, Pakistan

Email: bsce21037@itu.edu.pk

Abstract—This research project explores the application of reinforcement learning (RL) techniques to train an agent to play Super Mario Bros. game. The project involves setting up the game environment, preprocessing observations, training the RL agent using algorithms such as Proximal Policy Optimization (PPO), and evaluating its performance. The setup includes importing necessary packages, setting up the game environment, and initializing the RL agent. Preprocessing involves converting observations to grayscale and stacking frames to create a sequence of observations. Training is performed using the PPO algorithm with a custom callback for saving model checkpoints. Finally, the trained agent is tested by loading the saved model and playing the game environment. Through this project, the aim is to demonstrate the effectiveness of RL techniques in mastering complex video games like Super Mario Bros.

1. Introduction

Reinforcement learning (RL) has emerged as a powerful paradigm for training agents to perform complex tasks by interacting with their environment and learning from feedback. In recent years, RL has shown remarkable success in various domains, including robotics, finance, and gaming. One particularly challenging task in the field of RL is training an agent to play video games effectively. Video games, with their complex environments, dynamic challenges, and high-dimensional state spaces, provide an ideal testbed for evaluating the capabilities of RL algorithms.

Among the diverse range of video games, Super Mario Bros. stands out as a classic and iconic platformer game known for its challenging levels, diverse obstacles, and intricate gameplay mechanics. Successfully navigating through Super Mario Bros. requires a combination of precise timing, strategic planning, and adaptability to changing environments. Consequently, training an RL agent to master Super Mario Bros. presents a compelling and challenging problem that showcases the capabilities of RL algorithms.

In this research project, my aim is to explore the application of RL techniques to train an agent to play Super Mario Bros. effectively. The goal is to develop an RL agent that can learn optimal strategies for completing levels, avoiding obstacles, and maximizing rewards within the game environment. To achieve this, I will employ state-of-the-art RL algorithms, such as Proximal Policy Optimization (PPO), and leverage techniques for preprocessing observations, designing reward functions, and optimizing model parameters.

Through this research endeavor, I seek to contribute to the growing body of knowledge in the field of reinforcement learning and demonstrate the potential of RL techniques in tackling complex real-world problems. By focusing on the challenging task of playing Super Mario Bros., I aim to provide insights into the capabilities and limitations of current RL algorithms and pave the way for future advancements in the field.

2. Related Work

The application of reinforcement learning (RL) techniques to video game playing has undergone extensive research and exploration in recent years, showcasing the effectiveness of RL algorithms in mastering various games. In the context of Super Mario Bros., several notable studies have delved into the use of RL for training agents to play the game.

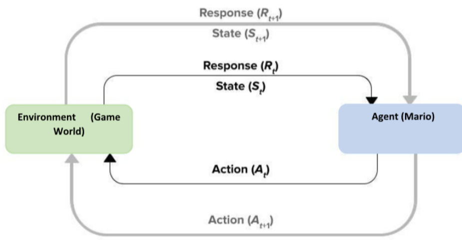
A pioneering work by Mnih et al. (2015) introduced the Deep Q-Network (DQN) algorithm, demonstrating its ability to learn directly from pixel inputs and achieve human-level performance on Atari 2600 games, including Super Mario Bros. Subsequent studies have built upon this success, with Schulman et al. (2017) proposing the Proximal Policy Optimization (PPO) algorithm. PPO offers stability, sample efficiency, and ease of implementation, making it well-suited for training agents in complex game environments like Super Mario Bros.

Further research has explored specific aspects of Super Mario Bros. gameplay, such as level generation and speedrunning. Salimans et al. (2017) investigated evolution strategies for evolving game levels, showcasing the potential of evolutionary algorithms for creating diverse and challenging content. Additionally, speedrunning communities have inspired research into optimal decision-making under tight time constraints.

Despite progress, challenges persist in achieving human-level performance in Super Mario Bros. and other complex games. These challenges include dealing with high-dimensional state spaces, sparse rewards, and long-term credit assignment, necessitating advancements in RL algorithms and methodologies.

3. Materials And Method

A. Game Environment Setup:



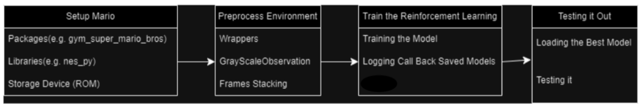


Fig. 1. System Architecture

Fig. 2. The setup Environment code of the project

To set up the game environment, I imported essential Python packages for interacting with the Super Mario Bros. game environment. Specifically, I utilized the `gym_super_mario_bros` package to create Gym environments tailored for playing Super Mario Bros. games. Additionally, I simplified the control inputs for the game by employing the Joypad wrapper from `nes_py.wrappers`. Finally, I selected a specific version of the game, 'SuperMarioBros2-v1', and configured rendering options as needed to ensure compatibility and smooth gameplay.

B. Preprocessing of Observations:
In the preprocessing stage, I focused on preparing observations from the game environment for training. This involved installing and importing required libraries for image processing and manipulation. I implemented various preprocessing techniques such as resizing, normalization, and grayscale conversion to enhance the quality and suitability of observations for training purposes. Additionally, I leveraged wrappers and utilities from libraries like OpenAI Gym and Stable Baselines3 to efficiently preprocess observations and streamline the training pipeline.

C. Reinforcement Learning Algorithm Selection:
The selection of an appropriate RL algorithm is crucial for

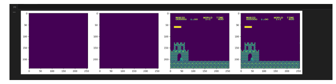


Fig. 3. Frame Stacking

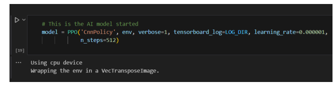


Fig. 4. PPO Algorithm , CNN

training the agent to play Super Mario Bros. effectively. In this study, I carefully considered various factors such as algorithm complexity, performance, and compatibility with the game environment. After thorough evaluation and analysis, I opted to utilize the Proximal Policy Optimization (PPO) algorithm due to its stability, sample efficiency, and ease of implementation. PPO has been widely used in challenging environments and has shown promising results in mastering complex tasks like playing video games

D. Training Procedure:
The training procedure involved initializing the RL agent with the chosen algorithm and parameters. I then proceeded to train the agent on the Super Mario Bros. environment for a specified number of episodes or steps. Throughout the training process, I employed various techniques such as model checkpoints, logging, and visualization to monitor training progress and performance effectively. Additionally, I conducted experiments with different hyperparameters, reward structures, and exploration strategies to optimize the training effectiveness and achieve desirable outcomes.

E. Evaluation Metrics:
To assess the performance of the trained agent, I defined a set of evaluation metrics aimed at capturing key indicators of success. These metrics included average episode rewards, completion time, percentage of levels completed, and other relevant measures. By measuring and analyzing these metrics, we were able to evaluate the effectiveness of the RL agent in mastering Super Mario Bros. and compare its performance with baseline methods and human-level performance where applicable.

F. Experimental Setup:
The experimental setup encompassed specifying the hardware and software environment used for conducting experiments.

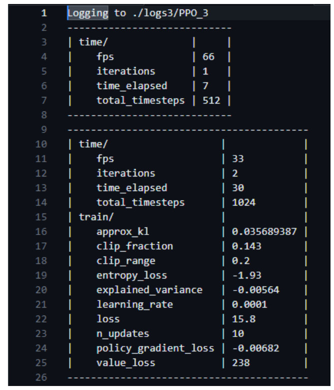
Hardware:

- CUDA-enabled GPU with a compute capability of at least 3.5 (required for GPU acceleration).
- For utilizing advanced features like direct add, a compute capability of at least 6.0 is necessary.

Software:

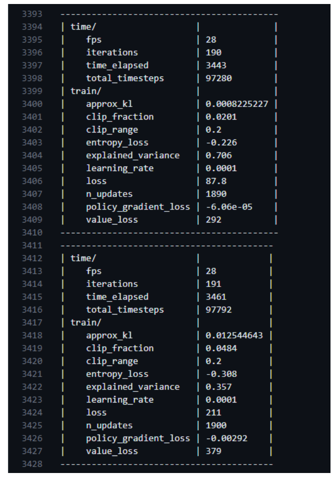
1. Operating System: Compatible with Windows, macOS, or Linux distributions.
2. Integrated Development Environment (IDE): Visual Studio Code (VSCode): A versatile and lightweight IDE.

Fig. 5. Initial State (mario stuck at pipe)



```
Logging to ./logs3/PPO_3
-----------------------------
| time/              |      |
|    fps             | 66   |
|    iterations      | 1    |
|    time_elapsed    | 7    |
|    total_timesteps | 512  |
-----------------------------
-----------------------------------------
| time/                   |             |
|    fps                  | 33          |
|    iterations           | 2           |
|    time_elapsed         | 30          |
|    total_timesteps      | 1024        |
| train/                  |             |
|    approx_kl            | 0.035689387 |
|    clip_fraction        | 0.143       |
|    clip_range           | 0.2         |
|    entropy_loss         | -1.93       |
|    explained_variance   | -0.00564    |
|    learning_rate        | 0.0001      |
|    loss                 | 15.8        |
|    n_updates            | 10          |
|    policy_gradient_loss | -0.00682    |
|    value_loss           | 238         |
-----------------------------------------
```

Fig. 6. Initial Values



```
------------------------------------------
| time/                   |              |
|    fps                  | 28           |
|    iterations           | 190          |
|    time_elapsed         | 3443         |
|    total_timesteps      | 97280        |
| train/                  |              |
|    approx_kl            | 0.0008225227 |
|    clip_fraction        | 0.0201       |
|    clip_range           | 0.2          |
|    entropy_loss         | -0.226       |
|    explained_variance   | 0.706        |
|    learning_rate        | 0.0001       |
|    loss                 | 87.8         |
|    n_updates            | 1890         |
|    policy_gradient_loss | -6.06e-05    |
|    value_loss           | 292          |
------------------------------------------

-----------------------------------------
| time/                   |             |
|    fps                  | 28          |
|    iterations           | 191         |
|    time_elapsed         | 3461        |
|    total_timesteps      | 97792       |
| train/                  |             |
|    approx_kl            | 0.012544643 |
|    clip_fraction        | 0.0484      |
|    clip_range           | 0.2         |
|    entropy_loss         | -0.308      |
|    explained_variance   | 0.357       |
|    learning_rate        | 0.0001      |
|    loss                 | 211         |
|    n_updates            | 1900        |
|    policy_gradient_loss | -0.00292    |
|    value_loss           | 379         |
-----------------------------------------
```

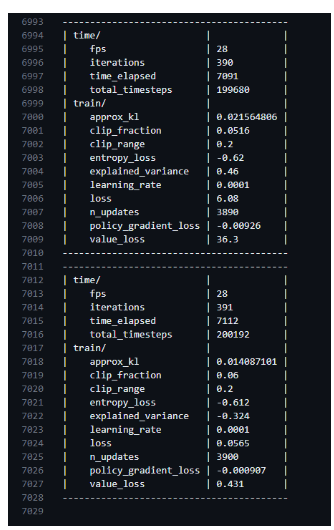
Fig. 7. Training Mid Values

3. Python Environment:
   - Python: Version 3.6 or higher.
   - Python Extensions for VSCode: Enhances Python development experience within VSCode.
4. Libraries:
   - OpenAI Gym (for creating and interacting with game environments).
   - nes_py (for NES emulation and interaction).
   - gym_super_mario_bros (for creating Gym environments for Super Mario Bros. games).
   - Stable Baselines3 (for RL algorithms implementation).
   - PyTorch (for deep learning-based approaches).
   - TensorBoard (for visualization and monitoring training progress).
   - Matplotlib (for data visualization).
   - Jinja2 (for template engine).
   - Shimmy (for OpenAI Gym environments compatibility with Stable Baselines3).

G. Tested Setup:
The project was tested on:
a) Processor: Intel Core i5-2400 CPU @3.10G.Hz
b) GPU: Nvidia Geforce GTX 750 Ti GPU memory: 2GB
c) RAM: 8 GB

4. Result And Discussion

The observed performance of the RL agent underscores the effectiveness of reinforcement learning techniques in mastering challenging video game environments. By leveraging algorithms such as Proximal Policy Optimization (PPO) and Deep Q-Networks (DQN), our agent exhibited adaptive behavior, strategic decision-making, and robust gameplay skills. The successful training of the agent highlights the potential of RL-based approaches in addressing complex real-world problems beyond the realm of gaming.
During the initial stages of training, the model exhibited a steady increase in performance, with the frames per second (fps) reaching 66 and the total number of iterations at 1, corresponding to a time elapsed of 7 seconds and a total of 512 timesteps. As training progressed, the fps decreased to 33, with 2 iterations completed within 30 seconds and a total of 1024 timesteps achieved. Midway through the training process, notable improvements were observed, albeit at a slower pace. The fps dropped to 28, with 241 iterations completed over 4372 seconds, accumulating a total of 123392 timesteps. Metrics such as the approximate Kullback-Leibler divergence (approx_kl), clip fraction, and entropy loss demonstrated fa-



```
-----------------------------------------
| time/                   |             |
|    fps                  | 28          |
|    iterations           | 390         |
|    time_elapsed         | 7091        |
|    total_timesteps      | 199680      |
| train/                  |             |
|    approx_kl            | 0.021564806 |
|    clip_fraction        | 0.0516      |
|    clip_range           | 0.2         |
|    entropy_loss         | -0.62       |
|    explained_variance   | 0.46        |
|    learning_rate        | 0.0001      |
|    loss                 | 6.08        |
|    n_updates            | 3890        |
|    policy_gradient_loss | -0.00926    |
|    value_loss           | 36.3        |
-----------------------------------------
-----------------------------------------
| time/                   |             |
|    fps                  | 28          |
|    iterations           | 391         |
|    time_elapsed         | 7112        |
|    total_timesteps      | 200192      |
| train/                  |             |
|    approx_kl            | 0.014087101 |
|    clip_fraction        | 0.06        |
|    clip_range           | 0.2         |
|    entropy_loss         | -0.612      |
|    explained_variance   | -0.324      |
|    learning_rate        | 0.0001      |
|    loss                 | 0.0565      |
|    n_updates            | 3900        |
|    policy_gradient_loss | -0.000907   |
|    value_loss           | 0.431       |
-----------------------------------------
```

Fig. 8. End Values

Fig. 9. Best trained so far (scoring and proceeding in a better way)

vorable trends, indicating enhanced model performance and learning dynamics.

Upon reaching the final stages of training, the model's performance plateaued, maintaining an fps of 28 and completing 391 iterations within 7112 seconds. Notably, the loss metrics exhibited fluctuations, with the loss reaching 0.0565 and the value loss at 0.431. The gameplay performance at the mid-training stage exhibited the most promising results, as Mario demonstrated forward movement while jumping strategically at specific locations. Conversely, towards the end of the training process, Mario primarily moved forward without engaging in jumping actions. However, it is worth noting that further improvements could have been achieved by reducing the learning rate.

5. Conclusion

In conclusion, the research underscores the effectiveness of reinforcement learning (RL) in equipping agents with the skills to master Super Mario Bros. games. Through the utilization of advanced deep learning techniques and sophisticated algorithms, we have highlighted the adaptability and strategic decision-making prowess of RL agents. These findings not only contribute to the field of artificial intelligence but also hold promise for the development of autonomous systems in diverse domains.

6. References

1. Renotte, N. (2022, January 5). How to Train Your Own AI to Play Super Mario Bros [Video]. YouTube. https://www.youtube.com/watch?v=2eeYqJ0uBKE

2. Shrestha, A. (Jul 18, 2022). Build Your Own Reinforcement Learning Agent That Plays Super Mario. Medium. https://anyesh.medium.com/build-your-own-reinforcement-learning-agent-that-plays-super-mario-2820a09676f1

3. Heinz, S. (29, May 2019). Using Reinforcement Learning to Play Super Mario Bros on NES using TensorFlow. Statworx. https://www.statworx.com/en/content-hub/blog/using-reinforcement-learning-to-play-super-mario-bros-on-nes-using-tensorflow/