CITS3001 Super Mario Project

Sijing Aloysius Loh 24028283
Olivia Morrision 23176135

For this project, we developed two AI agents to control Mario in the gym-super-mario-bros environment. The first agent uses a rule-based approach and uses OpenCV for image processing, taking actions based on these results. The second agent was trained on the Proximal Policy Optimisation algorithm, PPO, from Stable Baselines3. In this report, we attempt to find the optimal learning rate for the second agent and run multiple experiments to compare both agent's their performances, strengths and weaknesses.

Rule-Based Agent:

This algorithm uses a rule-based approach to make decisions in the game, and relies on OpenCV to process visual information from the game screen. Its objective is to navigate Mario to the end of each stage by avoiding enemies and jumping over obstacles. Templates of the game are assigned to their respective entities. This allows the agent to recognise objects in the game and choose actions based on the rules provided for those objects. A mask is created around each template which instructs the agent to ignore "sky" pixels in the template. This ensures that only the object is matched. The templates are also converted to grayscale to allow for easier matching in different levels where objects are of a different colour. This prevents the need for additional templates of different colours. For each object detected on screen, the agent will choose an action based on a set of predefined rules and heuristics. These rules encapsulate strategies for dealing with various scenarios such as jumping to avoid enemies.

The rule-based algorithm is highly transparent and interpretable. Rules and heuristics are explicit, making it easy to understand the agent's decision-making process. This strength is particularly useful for debugging as we can easily explain the agent's behaviour. It is also computationally efficient. The agent is able to begin playing with its defined rules, without having to spend any time or computational resources training or learning. This allows for rapid prototyping and experimentation where we can iteratively refine rules and instantly see the outcome of those refinements.

The rule-based algorithm lacks adaptability and learning capabilities. It does not store its past experiences and is unable to learn from them, making it struggle in the face of dynamic and unforeseen situations. This can be seen in the experiment on generalisation to unseen levels. As the game progresses, new and more dynamic obstacles appear, rendering the current rules ineffective. It also lacks exploration mechanisms. The agent strictly follows predefined rules, never exploring new actions or strategies. This hinders its ability to discover more optimal ways to progress in the game. If the rules do not allow for the agent to look for power-ups, the agent will never obtain a power-up which might allow it to progress faster.

PPO Agent:

Background to Policy Gradient Methods

A "subset" of Policy based methods are Policy-Gradient methods, which optimise the policy using gradient ascent to find parameters that maximise expected return (Simonini, 2018). This method employs a stochastic policy, a probability distribution of actions, to account for uncertainty in a given state (Baeldung, 2023). However, this method faces many issues, including (Schulman et al., 2017) instability due to large policy updates which commonly miss high rewards, sensitivity to hyperparameters, prone to getting stuck in local optima and sample inefficiency as samples are only used once (Van Heeswijk, 2022).
Trust Region Policy Optimisation was developed to address these issues by introducing a "surrogate objective function" which is maximised, and uses "trust-region constraints" to constrain the size of the policy update (Karunakaran, 2020). While TRPO provides more stability by avoiding large policy updates, it is much more difficult to implement and fine tune (Schulman et al., 2017) and has a significantly slower updating procedure (Van Heeswijk, 2022).

Introduction to PPO

Proximal Policy Optimisation (PPO), introduced in 2017 by OpenAI, is a policy gradient method which was designed to mitigate issues with stability, implementation difficulties, data efficiency and hyperparameter tuning that other methods faced (Schulman et al., 2017).
PPO aimed to provide an improvement on Trust Region Policy Optimisation (TRPO) by using a similar idea to constrain policy updates while still searching for an optimal policy with higher rewards. PPO introduces a "clipped surrogate objective function" which limits the policy update to a small range (between 0.8 and 1.2), to ensure that the new policy does not stray too far from the old policy.
The Clipped Surrogate Objective Function relies on two functions:

- A ratio function, which indicates if an action is more or less likely in the new policy than the old policy.
- An advantage function, which estimates the expected improvement of taking an action compared to the value of the average action in a given state.

The Objective function then takes the minimum of the unclipped ratio and the clipped ratio multiplied by the advantage function. By taking the minimum, large policy updates are penalised (Simonini, 2018), ensuring that it does not deviate too far from the previous policy and that the updated policy is not too heavily influenced by this single calculation (Arxiv Insights, 2018).

Stable Baselines3 PPO Algorithm and Implementation

We implemented PPO using Stable Baselines3 to train our second agent, following Nicholas Renotte's tutorial, "Build an Mario AI Model with Python". Stable Baselines3 (SB3) documentation states that their implementation of PPO follows the "Actor-Critic Style" which combines value based and policy-based methods by using an "actor" to control an agent's behaviour using a policy-based method and a "critic" that judges the quality of that action using a value-based method. SB3 implements both the actor and the critic as neural networks. This approach is demonstrated in the pseudo code below:

In each training iteration, the actor begins by collecting data through interacting with the environment guided by the policy for a specified amount of timesteps (note in our implementation, only one actor is used). The results are returned as state-action pairs, with the resulting rewards obtained from the environment. The critic then estimates the state values and advantages are computed to determine how much "better" or "worse" the action was in a given state than it was predicted/estimated to be (Trivedi, 2019).
SB3 implements both the actor and the critic using neural networks. This approach requires an overall loss function which is the combination of:

- The policy loss, calculated by the clipped surrogate objective function. This is maximised to encourage the actor to seek higher rewards.
- The value loss, calculated by the value function (critic). This reflects the difference between the predicted and target state values and is minimised to encourage the critic to make accurate state value predictions/estimates.
- The entropy bonus, which is an incentive that introduces randomness to avoid local optima and encourage exploration.

The result is what guides the agent's training and is used to update actor/critic network parameters as it reflects the agent's performance, accuracy and randomness/certainty (Bick, 2021). This function is optimised to encourage actions that result in higher rewards and exploration whilst also ensuring that the value estimation is accurate.

Training Process (Pre-processing/wrappers and Policy)

To train our agent using the PPO algorithm from SB3 in the Gym Super Mario Bros environment, the environment was pre-processed with the following wrappers to optimise training:

- "GreyScaleObservation" converted the observation space of the environment from RGB to greyscale to reduce the complexity, storage required and overall time taken as this reduced the observation/state space from 3 channels to 1.
- "DummyVecEnv" converted the environment to a simple vectorized environment to make it compatible with the algorithm
- "VecFrameStack" which "stacks" four consecutive frames together to enhance learning and training stability by capturing movement, Mario's actions, enemies and other features of the environment.

Given that the agent would be training in an environment that returned observations as images, the PPO "actor" was implemented using SB3's Convolutional Neural Network (CNN) policy. The CNN policy is able to process the stack of images as input and extracts data through convolutional and pooling network layers. This extracted data is then mapped to a probability distribution of actions through fully connected layers (IBM, n.d.). The output is the stochastic policy (probability distribution of taking an action in a given state) (Baeldung, 2023).

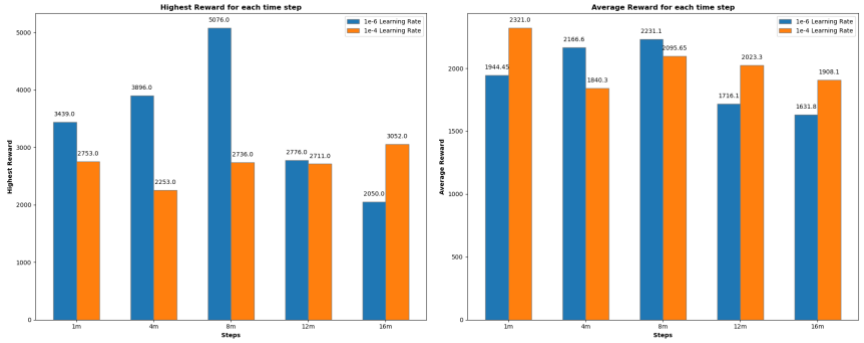
Training Results (and Experiments)

Our first experiment was to find the appropriate learning rate to train the PPO agent which would result in a high performing agent. The learning rate is a key hyperparameter which controls the extent to which the model is adjusted in response to the loss function during each update of the model weights. it directly impacts the speed at which the model adapts and converges to reach stability (Zulkifli, 2018).

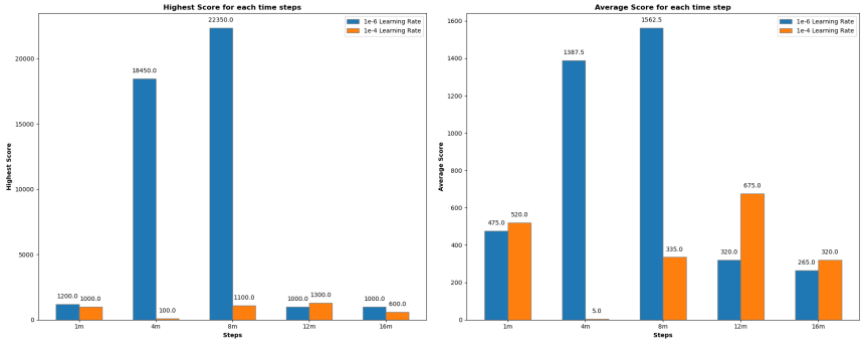
We experimented with two different learning rates: 1e-4 and 1e-6. For both learning rates, we kept the other hyperparameters the same to ensure consistency by using the default hyperparameters provided by SB3 and trained both models for 16 million steps to allow learning development.

Experiments

Both the 1e-6 LR and 1e-4 LR models were tested on the first stage of the first world at different timesteps to find the best performing model in terms of reward and in-game score.



Comparing models based on rewards, the 1e-6 LR model had a significant rewards drop after 8 million steps, while the 1e-4 model maintained a higher average reward of 2037.4 across all steps, outperforming the 1e-6 model's 1938.01 average. Though the 1e-4 model had a peak average reward at 1 million steps, this model's highest reward was only 2753. The best model was the 1e-6 at 8 million steps with an average reward of 2231.1 and a peak of 5076, suggesting consistent high rewards.



When assessing the scores, a dramatic difference was shown between the 1e-6 model at both 4 million and 8 million steps in comparison to other models. Both models show very inconsistent

scores, with the 1e-4 model at 4 million steps not being able to get a higher score than 100 across all 20 runs despite all other 1e-4 models getting at least an average score of 300.
We concluded that our best model from this experiment was the 1e-6 model at 8 million steps, as this model received the overall best results for both rewards and score. These experiments showed us the instability of both models as demonstrated by the inconsistency when assessing scores and most reward results indicating most did not make it past the first level, as even our best model from this experiment did not make it past the first stage every single run. This was unexpected as both agents were trained for a long time. This was the model that we used in further experiments against our first agent.

PPO Strengths and Weaknesses

The decision to implement PPO was made due to its reputation and well-known strengths, including simplicity, stability, and robustness with hyperparameter tuning, when compared to other policy-gradient methods. However, we found that PPO performed well below our expectations and encountered challenges with stability. Despite training two models with different learning rates for 16 million steps (which took multiple days), we found that we would require more hyperparameter tuning to find an optimal model.

Comparisons Between Agents:

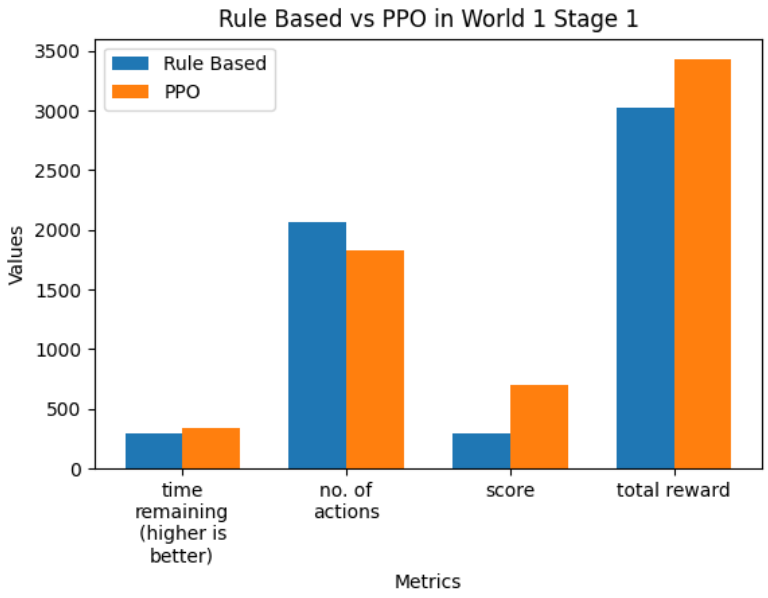
Performance in World 1 Stage 1:

In the first experiment, we compared the performance between the rule-based and PPO agents in World 1 Stage 1 of the game. This is the first level of the game and the one that both agents were trained on. Both agents played the first level to completion 5 times and we measured their performance based on an average of the following metrics:

- time remaining: the amount of time left on the clock when Mario reaches the flag where a higher value means better performance
- score: the in-game score achieved by the agent at the end of the level
- total reward: the sum of the reward function throughout the level
- number of actions: the number of actions the agent takes to complete the level

We observed that the PPO agent performed more favourably than the rule-based agent on all of the above metrics. It completed the level faster with a lower number of actions while achieving a higher score than the rule-based agent. This is likely due to the primitive nature of the rule-based agent. It has a fixed set of rules which it follows strictly, while the PPO agent is able to discover more optimal routes and increase its reward function.

Where the rule-based agent lacked in the aforementioned metrics, it made up for with its consistency. Given the same environment, the rule-based agent always executes the same action at the same place and time. As such, it was able to consistently beat the first level every try, taking the same path for each run. There was no need to take an average of the metrics since the values are the same each time. The PPO agent struggled to complete the first level consistently, only managing to do so less than 20% of the time.



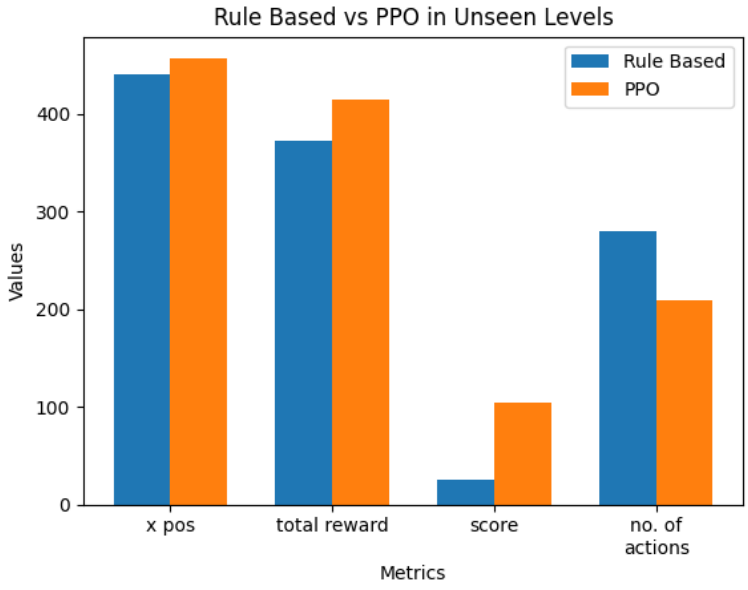
Generalisation to unseen levels:

For the second experiment, we compared the performance of the rule-based and PPO agents on levels that they were not trained on. The levels are: Stage 1, 2, 3, 4 of World 4 and Stage 1, 2, 3, 4 of World 8. Both agents played each stage 5 times and the runs where the agent got the furthest were taken into account. Performance was measured based on the average values of the following metrics for the 8 stages:

- x pos: how far right the agent gets in the game
- total reward: the sum of the reward function throughout the run
- score: the in-game score achieved by the agent at the end of the run
- number of actions: the number of actions the agent takes throughout the run

Similar to the first experiment, the PPO agent outperformed the rule-based on all of the above metrics. On average, the PPO agent managed to go further into the unseen levels than the rule-based agent, achieving a higher total reward and score with fewer actions. This was to be expected since most of the rules for the rule-based agent are specific to World 1 Stage 1 and a bit of stage 2. As such, it performs consistently well there, but when introduced to a new level with new obstacles and enemies, it is unable to detect and overcome them.

Conversely, the PPO agent learns to maximise the reward function through interaction with the game environment. It does not rely on predetermined rules but instead uses a policy that can adapt to new situations to maximise the reward. This allows the PPO agent to perform better than the rule-based agent in unseen levels.



Visualisation techniques:

Rule-Based Agent:

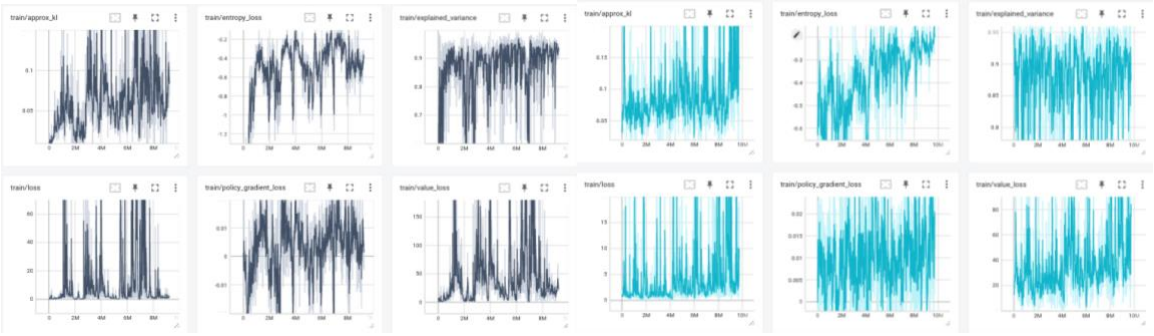
For the rule-based agent, a window is created using openCV which shows the current game frame and bounding boxes around objects that are detected. It provides a real-time view of what the agent detects in the game at any moment, which allows us to observe the agent's decision making process visually. It also verifies that the agent is accurately recognising in-game objects. Different colours are used for the bounding box of each object to ensure that the objects are being identified correctly.

PPO Agent:

We used Stable Baselines3's TensorBoard integration to visualise the training progress. Due to hardware constraints, the 1e-4 model was trained in intervals of 9 and 6 million steps and the 1e-6 learning rate model was trained at 1 million steps intervals so our visualisation was split across multiple TensorBoard logs. Only a few 1e-6 logs have been selected for demonstration purposes.

The loss plots for the 1e-4 model show that the model had some stability around 2m, 5m and 9m steps however after 10 million it began to diverge and become unstable, indicating the learning rate was too high (AurelianTactics, 2018). However, the plots for the 1e-6 model showed much more instability across all model intervals as overall loss was much higher and diverged a lot more than the 1e-4 model. Both models showed great instability, indicating that both learning rates were not appropriate. Further experimentation with a learning rate scheduler is recommended (Jason Brownlee, 2020).

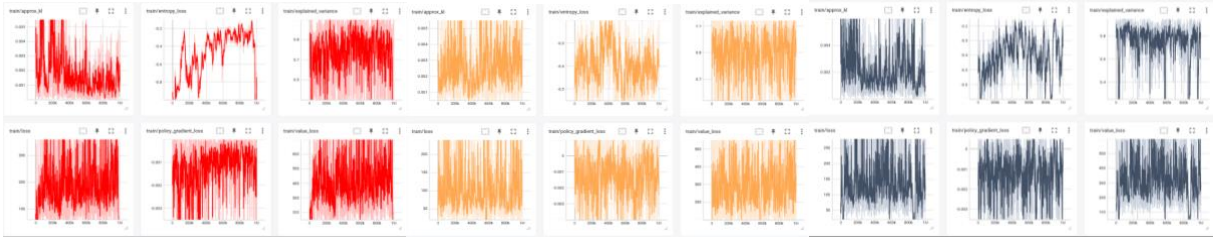
TensorBoard Plots for Training Model at 1e-4 LR



0-9 million steps

9-16 million steps

TensorBoard Plots for Training Model at 1e-6 LR



0-1 million steps

7-8 million steps

15-16 million steps

References

Arxiv Insights (2018). *An introduction to Policy Gradient methods - Deep Reinforcement Learning*. [online] YouTube. Available at: https://www.youtube.com/watch?v=5P7I-xPq8u8&ab_channel=ArxivInsights [Accessed 16 Oct. 2023].

AurelianTactics (2018). *Understanding PPO Plots in TensorBoard*. [online] Medium. Available at: https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2.

Baeldung (2023). *Deterministic vs. Stochastic Policies in Reinforcement Learning*. [online] Baeldung. Available at: https://www.baeldung.com/cs/rl-deterministic-vs-stochastic-policies.

Bick, D. (2021). *Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization*. [Thesis (Master's Thesis / Essay)] *fse.studenttheses.ub.rug.nl*, pp.3–28. Available at: https://fse.studenttheses.ub.rug.nl/id/eprint/25709.

IBM (n.d.). *What are Convolutional Neural Networks? | IBM*. [online] IBM. Available at: https://www.ibm.com/topics/convolutional-neural-networks.

Jason Brownlee (2020). *Understand the Impact of Learning Rate on Neural Network Performance*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

Karunakaran, D. (2020). *Trust Region Policy Optimisation(TRPO) — a policy-based Reinforcement Learning*. [online] Intro to Artificial Intelligence. Available at: https://medium.com/intro-to-artificial-intelligence/trust-region-policy-optimisation-trpo-a-policy-based-reinforcement-learning-fd38ff9e996e.

Renotte, N. (2021). *Build an Mario AI Model with Python | Gaming Reinforcement Learning*. [online] YouTube. Available at: https://www.youtube.com/watch?v=2eeYqJ0uBKE&ab_channel=NicholasRenotte.

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

Simonini, T. (2018). *Introducing the Clipped Surrogate Objective Function - Hugging Face Deep RL Course*. [online] huggingface.co. Available at: https://huggingface.co/learn/deep-rl-course/unit8/clipped-surrogate-objective.

Trivedi, C. (2019). *Proximal Policy Optimization Tutorial (Part 2/2: GAE and PPO loss)*. [online] Available at: https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-2-2-gae-and-ppo-loss-fe1b3c5549e8.

Van Heeswijk, W. (2022). *Proximal Policy Optimization (PPO) Explained*. [online] Towards Data Science. Available at: https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b#:~:text=Due%20to%20the.

Zulkifli, H. (2018). *Understanding Learning Rates and How It Improves Performance in Deep Learning*. [online] Medium. Available at: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.