Autonomous and Adaptive Systems Project Report

Mauro Dore
Alma Mater Studiorum,
University of Bologna
mauro.dore@studio.unibo.it

Abstract

This study investigates the application of the **Proximal Policy Optimization (PPO)** algorithm within the **ProcGen** environment - a suite of 16 procedurally generated game-like environments designed to benchmark both sample efficiency and generalization in reinforcement learning. Providing a detailed description of the PPO approach, including its development and implementation choices. The experimental results demonstrate that the PPO approach significantly outperforms a random agent, that serves as baseline for evaluation.

1 Introduction

The aim of the *Autonomous and Adaptive Systems* course project is to create and evaluate the performance of a Reinforcement Learning (RL) model in the *ProcGen* environment developed by OpenAI. The project explores how agents learn and adapt to various scenarios.

The ProcGen environment [4], provides a standardized evaluation of the agent's ability to generalize between different scenarios within the same game configuration. This environment is designed to challenge the agent's ability not only to learn fixed paths or sequences, but to develop a robust strategy that adapts to new and unexpected game configurations.

The subset of ProcGen games used during the project includes **Coinrun** and **Caveflyer**. These environments have different characteristics and, consequently, require different skills to be solved. They were chosen to demonstrate how the developed algorithm is able to learn these skills and outperform a random agent used as a baseline.

Among the various Actor-Critic algorithms, *Proximal Policy Optimisation* (PPO) was selected for this study. This choice was complemented by the integration of the *IMPALA* network architecture and the *Generalised Advantage Estimation* (GAE) for advantage calculation.

The evaluation of this project is reflected not only in the performance of the developed model, but also in the analysis of the design choices, and decisions made during the development process. This report attempts to provide an in-depth narrative of the methodologies, experiments and results, contributing to the final result.

2 System description

In order to accomplish the project's objective, an *Actor-Critic* algorithm is considered a robust approach that combines the characteristics of both policy-based and value-based methodologies.

Following an observation, the data is input into a *Convolutional Neural Network* (CNN) model. The organization of this model's pipeline is as follows:

- At time t, an action A_t is sampled based on a policy $\pi(a|S_t, \theta)$ from a discrete action space comprising 15 possible choices.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

- Simultaneously, a value $\hat{v}(S', w)$ provides an estimation of the desirability of being in state S_t given the current policy.
- The action A_t is executed for n steps (frame skip), and the tuple of the type $(S_t, A_t, P(A_t), \hat{v}(S_t, \theta), R_t, done)$ from the first step of the frame skip is stored.
- Steps 1-3 are repeated until a batch size number of tuples is collected. These tuples are shuffled to remove correlation, divided into mini-batches, and fed to the model, which is trained for n epochs
- The loop restarts with the updated model.

The structure above contributes to the robustness and efficiency of the proposed solution, while the *frame skip* helps accelerate the process.

2.1 Model

The model, as illustrated in Figure 1, is implemented using IMPALA [5] as a reference. It comprises a shared Convolutional Neural Network (CNN) and two distinct heads responsible for generating policy and value outputs. The paper [6] was also consulted for additional suggestions and modifications to the original architecture, and as recommended, the LSTM layer was removed due to its negligible contribution to learning.

The *Convolutional layers* are essentials for capturing spatial features from the input, and the *Dense* layer discerns complex feature interactions. The *Actor* layer outputs a probability distribution from 15 logits passed through a SoftMax activation function. The *Critic* layer outputs a single value via a linear activation function.

Residual Blocks [2], with skip connections, are key for maintaining gradient effectiveness, mitigating the vanishing gradient problem, enabling efficient training of deeper networks, and speeding up convergence. Given the large data volumes from complex environment interactions in this project, Residual Blocks prove especially beneficial.

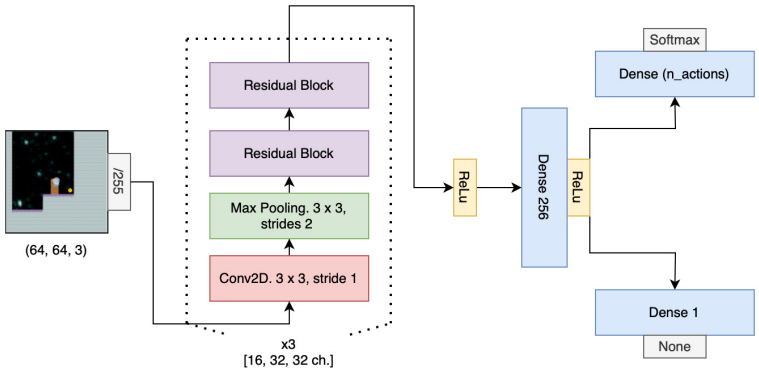

Figure 1: Impala Network

Despite the complexity of hyperparameter selection in IMPALA, it demonstrated superior processing speed compared to other tested models. This made it a viable choice for this project.

The legacy version of the *Adam optimizer* was employed, which is expected to yield better performance on MacBooks equipped with M1 Silicon chips.

2.2 Proximal Policy Optimization

Proximal Policy Optimization aims to make incremental updates to the policy, avoiding excessive deviation from the original policy.

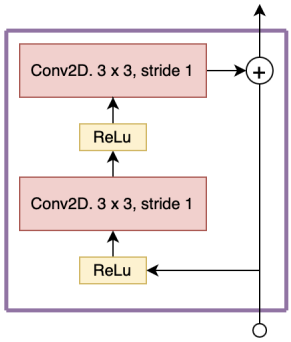

Figure 2: Residual Block

This is achieved through the introduction of an objective function [2], denoted as *Lclip*, which limits the variation of the policy at each update. The clipping operation in Lclip takes the ratio of the current policy to the old policy and selects the minimum of the two, i.e., the clipped and unclipped ratios.

The loss Lclip is given by:

$$\hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \tag{1}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2}$$

in which ϵ is an hyperparameter which define how far the new policy is allowed to go from the old.

The *entropy* is calculated using the Shannon entropy formula, which sums the product of probabilities and their logarithms across possible actions.

$$Entropy = -\sum p(x) \log p(x) \tag{3}$$

The final objective contains 3 parts, first one is Lclip, the second one is MSE of the critic network (i.e. squared loss of state values predicted and target) and the third part is entropy.

$$L^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L^{\text{CLIP}}(\theta) + c_1 L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{4}$$

This mechanism ensures a balance between exploration and exploitation, leading to more stable and efficient learning.

2.3 Generalised Advantage Estimation

To compute the advantages of a policy within the algorithm the *Generalised Advantage Estimation* (GAE) was used [3]. GAE effectively balance the trade-off between bias and variance of advantage estimates by employing a weighted average of advantages calculated over varying time lengths.

Estimates of advantage in GAE are calculated as follows:

- $\hat{A}_t^{(1)}$ is high bias, low variance, while $\hat{A}_t^{(\infty)}$ is unbiased, high variance.

$$\begin{aligned}
\hat{A}_t^{(1)} &= r_t + \gamma V(s_{t+1}) - V(s) \\
\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s) \\
&\vdots \\
\hat{A}_t^{(\infty)} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V(s)
\end{aligned}$$

CoinRun				
Agent	Number of Episodes	Environment Steps	Average Return	Episode Lenght
PPO	507	125000	5.10	246.69
Random	180	125000	2.22	691.29

Table 1: Results obtained by Coinrun on the test environment

Where γ is the discount factor, r_t is the reward at time t, $V(s)$ is the estimated value function for state s, and k denotes the number of steps considered in the calculation.

- A weighted average of $\hat{A}_t^{(k)}$ it taken to balance bias and variance.

$$\hat{A}_t = \hat{A}_t^{GAE} = \frac{\sum_k w_k \hat{A}_t^{(k)}}{\sum_k w_k}$$

In this, w_k is a weight given to each term, chosen as $w_k = \lambda^{k-1}$ for some $\lambda \in [0, 1]$, which determines how quickly the weights decrease for estimates over longer time intervals.

- Calculation of $\hat{A}_t$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

3 Experimental setup and results

This section presents configurations and results from a subset of environments provided by Procgen, compared to a baseline random agent.

Most of the parameters were set based on the suggestions of the paper cited above, however the best values of learning rate, batch sizes and number of frame skip were identified through a grid search. The gym environments were configurated with distribution mode easy, background disabled and a fixed seed was used to ensure reproducibility.

The training was conducted on 200 levels, after initial tests showed the algorithm's superior performance on a single level. Testing for both the PPO Agent and the Random Agent was performed across an unlimited number of levels.

The algorithm's performance was evaluated using three metrics: *Number of Episodes*, *Average Return*, and *Episode Lengths*. Due to system limitations, the training of all games were run for a fixed 500K *Environment Steps*. The distribution of Environment Steps between training and testing was in an 80:20 ratio.

3.1 Coinrun

In this environment, a high score and a short episode duration is a strong indicator of performance.

Examining the training graph shown in Figure 3, it is observed that despite the variability, a positive trend is observable. This suggests that, nevertheless the fluctuations, the overall strategy yields improvements.

During the testing phase Table1, the algorithm demonstrated superior performance compared to the random agent, achieving an average return that was more than double. This improvement was observed in both the number of episodes and their duration, underscoring the efficiency of the algorithm. In particular, the agent required fewer steps to reach the goal compared to the baseline, indicating enhanced effectiveness.

3.2 Caveflyer

Also in Caveflyer environment, achieving a high score in a reduced episode duration highlights good performance.

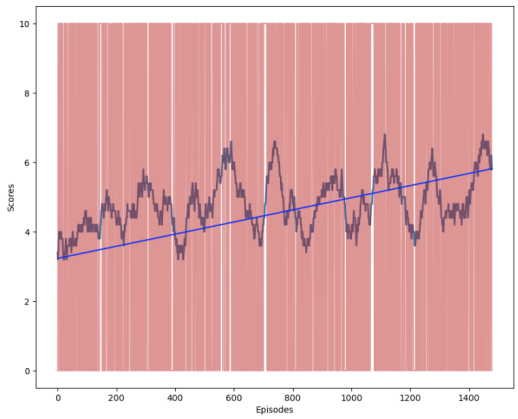

Figure 3: Training plot Coinrun 500000 timesteps

Caveflyer				
Agent	Number of Episodes	Environment Steps	Average Return	Episode Lenght
PPO	326	125000	3.74	383.98
Random	233	125000	2.72	540.13

Table 2: Results obtained by Caveflyer on the test environment

Looking at the training graph depicted in Figure 4, a positive trend is observed, indicating improvements as the number of games played increased. As shown in Table 1, the PPO agent outperformed the random agent in every metric, achieving a higher average return and a shorter episode length.

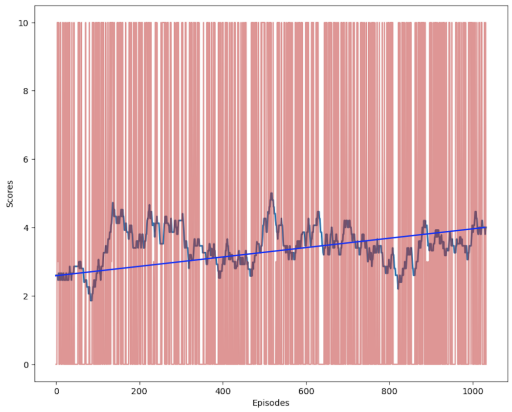

Figure 4: Training plot Caveflyer 500000 timesteps

4 Discussion

In the initial stages of the research, various *Actor-Critic* algorithms were tested. However, *instability* was encountered in the policy updates, leading to suboptimal policy modifications.

Data inefficiency was also encountered, related to the learning methods employed, which slowed down the learning process. This was probably because the neural networks require a substantial volume of data for effective learning.

One of the significant challenges that impacted the obtained results was the *computational architecture* used for running the code. The absence of a GPU significantly increased the code execution time.

Alternative platforms, such as Google Colab, were explored. However, the extensive model training times surpassed the allowable usage time of these platforms, making them impractical for use.

These challenges led to the implementation of Proximal Policy Optimization (PPO) in the research. PPO, thanks to it's multiple epochs minibatch updates and its objective function, works well in complex enviroment like those proposed by ProcGen, and offers solutions to the aforementioned issues, enhancing the stability of policy updates and improving data efficiency, thereby paving the way for a more effective and efficient learning process.

Regarding to the scores in the games, tests were also conducted with a different learning rates. For example, it was observed that the average score in the game CoinRun, passing from a learning rate of $5e^{-4}$ to $1e^{-4}$, can decrease the by one point.

Utilizing small mini-batch sizes in the algorithm resulted in minimal learning due to insufficient data. Conversely, an excessively large mini-batch size decelerates the learning process significantly. A similar pattern was observed when altering the number of frame skips.

5 Conclusion

Upon analysis of the results, the PPO-based agent outperformed the baseline. The disparity between the returns of the baseline and the agent was more pronounced in the CoinRun environment, probably due to a simpler strategy required for resolution, while Caveflyer necessitates of a more complex one. The models primarily exhibited two limitations. The first concerns to the challenge in identifying a strategy capable of maximizing the episodes scores. The second limitation was the restricted number of training steps that could be executed.

Nonetheless, the upward trajectory observed in the graphs suggests that training over a larger number of timesteps can potentially produce a higher scores than training over 500k timesteps."

About future improvements, in addressing slow learning, one potential solution is the deployment of multiple agents across parallel environments. This approach allows the simultaneous collection of multiple experiences, accelerating the learning process. However, the addition of more agents also increases the computational complexity.

Improvements to the clipping function could also be considered. For instance, asymmetric clipping with different limits for increases and decreases in the probability ratio. Alternatively, as proposed in [1], an adaptive clipping mechanism could be employed within a new surrogate learning objective. This could lead to a novel algorithm that optimizes policies based on a theoretical target for adaptive policy improvement.

Lastly, the integration of Prioritized Experience Replay (PER) could further enhance the algorithm's focus on experiences that are most relevant for policy updates.

References

[1] Mengjie Zhang Gang Chen, Yiming Peng. An adaptive clipping approach for proximal policy optimization, 2018.

[2] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms, 2017.

[3] Sergey Levine Michael Jordan Pieter Abbeel John Schulman, Philipp Moritz. High-dimensional continuous control using generalized advantage estimation, 2015.

[4] Jacob Hilton John Schulman Karl Cobbe, Christopher Hesse. Leveraging procedural generation to benchmark reinforcement learning, 2020.

[5] Remi Munos Karen Simonyan Volodymir Mnih Tom Ward Yotam Doron Vlad Firoiu Tim Harley Iain Dunning Shane Legg Koray Kavukcuoglu Lasse Espeholt, Hubert Soyer. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.

[6] Adrien Gaidon Andrey Kolobov Blake Wulfe Dipam Chakraborty Gražvydas Šemetulskis João Schapke Jonas Kubilius Jurgis Pašukonis Linas Klimas Matthew Hausknecht Patrick MacAlpine Quang Nhat Tran Thomas Tumiel Xiaocheng Tang Xinwei Chen Christopher Hesse Jacob Hilton William Hebgen Guss Sahika Genc John Schulman Karl Cobbe Sharada Mohanty, Jyotish Poonganam. Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark, 2021.