Autonomous and Adaptive Systems Project Report

Gian Mario Marongiu
Alma Mater Studiorum, Bologna
gianmario.marongiu@studio.unibo.it

Abstract

The Procgen Benchmark [6] is a suite of 16 procedurally generated, game-like environments designed by a team at OpenAI to assess both sample efficiency and generalization in reinforcement learning. This report explores the implementation of a Proximal Policy Optimization (PPO) approach [7] used to solve a subset of these tasks. This method aims to closely resemble the original approach described by OpenAI [4]. Specifically, this report describes the development of these experiments and the rationale behind the implementation choices, which sometimes differed from those in the original Procgen publication. The results obtained demonstrate a good tendency of the agent to learn and generalize, with outcomes that closely align with those described in the original work.

1 Introduction

The games tested are *fruitbot* and *coinrun* due to their steep learning curves, which ensure evident improvements even with a short training time. The actor-critic PPO algorithm proposed by OpenAI, and replicated in this study, is used for these experiments due to its reliability. PPO effectively reduces variance by combining value-based and policy-based methods, and its ability to prevent large policy updates improves stability. Additionally, PPO is highly scalable and performs well in high-dimensional, complex environments like those in the Procgen suite. Its efficiency is further enhanced by multiple epochs of mini-batch updates, allowing for better data retrieval from observations [7].

2 System Description

The computation revolves around a single Convolutional Neural Network (CNN) that functions as both the actor and critic. The choice to share parameters between the actor and critic is primarily motivated by performance considerations, particularly in terms of computational efficiency. This network elaborates information from $k = 16$ different parallel environments. During each training loop:

1. At time t, an action $A_{t,k}$ for each environment k is sampled based on a policy $\pi(a|S_{t,k}, \theta)$ from a discrete action space consisting of 15 possible choices.
2. At time t, a value $\hat{v}(S_{t,k}, \theta)$ provides an estimate of the desirability of being in state $S_{t,k}$ given the current policy.
3. The action $A_{t,k}$ is executed, the state is updated, and the trajectories relative to the current timestep are collected.
4. Steps 1-3 are repeated until a set number of trajectories is collected. These trajectories are shuffled to break the correlation introduced by their sequential nature, and the model is updated with three epochs of training using the formulas described in paragraph 2.2.
5. The loop restarts with the updated model.

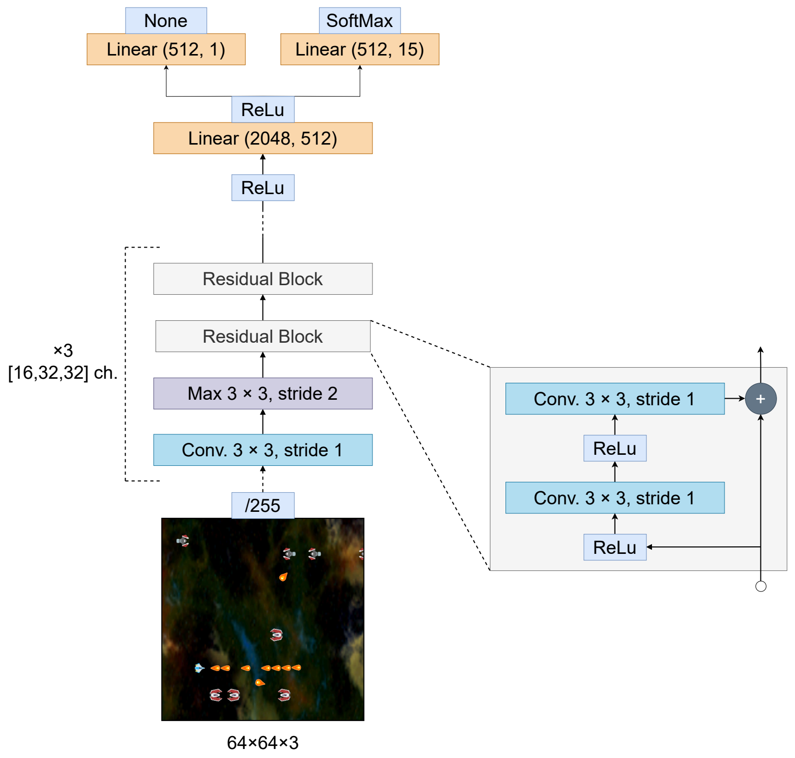

Figure 1: Architecture of the IMPALA-inspired model (1.15M parameters). The critic head (top-left) outputs a single logit, the actor head (top-right) outputs a probability distribution over the action space.

The trajectory collected at time t from the environment k is presented as a tuple of the form $(S_{t,k}, A_{t,k}, P_k(A_{t,k}), R_{t+1,k}, \hat{v}_k(S_t, \theta), done_k)$.

2.1 Model

The model, illustrated in Figure 1, draws inspiration from the original implementation of the IMPALA large model [5]. It has been modified by increasing the size of the linear layers and removing the Long Short-Term Memory (LSTM) layers, as OpenAI's experiments [4] demonstrated that LSTM layers do not yield significant improvements. Additionally, batch normalization is applied after each convolutional layer to enhance stability. The model has several key components: the convolutional layers are essential for extracting spatial features from the input observations; the residual blocks help mitigate the vanishing gradient problem and enable the model to learn deeper representations without performance degradation; the dense layer captures complex relationships and interactions between the extracted features.

The Actor component generates a probability distribution $\pi(a|S_{t,k}, \theta)$ by outputting 15 logits which are then passed through a SoftMax activation function. The Critic component produces a single value $\hat{v}(S_{t,k}, \theta)$, which is not further processed by any activation function.

The model weights are initialized using Xavier initialization, as done in OpenAI's experiments. The last policy layer, however, is initialized with weights that are 100 times smaller, as suggested in [1].

2.2 Loss Function and Weights Update

The PPO loss function [7] for the Actor model is defined as:

$$L^{CLIP}(\theta) = -\mathbb{E}_t \left[\min \left(r_{t,k}(\theta) \hat{A}^{GAE}_{t,k}, \text{clip}(r_{t,k}(\theta), 1-\epsilon, 1+\epsilon) \hat{A}^{GAE}_{t,k} \right) \right] \quad (1)$$

where

$$r_{t,k}(\theta) = \frac{\pi_\theta(a_{t,k}|s_{t,k})}{\pi_{\theta_{\text{old}}}(a_{t,k}|s_{t,k})} \tag{2}$$

is the probability ratio of the performed actions $a_{t,k}$ according to the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$. The function $\text{clip}(r_{t,k}(\theta), 1-\epsilon, 1+\epsilon)$ in equation 1 clips the probability ratio to the interval $[1-\epsilon, 1+\epsilon]$ to avoid large policy updates across sequential epochs of training.

$\hat{A}^{GAE}_{t,k}$ is the generalized advantage estimator (GAE) computed, before the data shuffling, independently for each environment as a sum of discounted TD errors δ [8]:

$$\hat{A}^{GAE}_{t,k} = \sum_{h=0}^{T} (\gamma\lambda)^h \delta_{t+h,k} \tag{3}$$

Where γ is the reward discount factor and the parameter λ controls the trade-off between bias and variance in advantage estimates, introduced by the discount factor. When λ is larger, the weight of the n-step return increases, because of its smaller bias and larger variance, the corresponding advantage estimator also has a smaller bias and larger variance. On the contrary, with a smaller λ, the variance is small, and the bias is large [3]. Their value is set to a fixed amount empirically determined by OpenAI. In particular, $\gamma = 0.999$ because, despite the episodic nature of the tasks, a discount factor very close to 1 may benefit the training process in environments with long episodes.

The TD-error $\delta_{t,k}$ in equation 3 is computed as follows:

$$\delta^V_{t,k} = R_{t,k} + \gamma\hat{v}(S_{t+1,k}, \theta) - \hat{v}(S_{t,k}, \theta) \tag{4}$$

The value function loss is computed as:

$$L^{VF}(\theta) = \mathbb{E}\left[\left(\hat{v}(S_{t,k}, \theta) - (\hat{A}^{GAE}_{t,k} + \hat{v}(S_{t,k}, \theta_{old}))\right)^2\right] = \mathbb{E}\left[\left(\hat{v}(S_{t,k}, \theta) - \hat{v}^{target}_{t,k}\right)^2\right] \tag{5}$$

Where the target $\hat{v}^{target}_{t,k}$ replaces the classic discounted rewards commonly used to update the critic. This approach reduces the high variance associated with long-term empirical returns, providing a more stable and accurate target for value function updates [2].

These loss functions are combined into a single formula $L^{total}(\theta)$, defined as:

$$L^{total}(\theta) = L^{CLIP}(\theta) + c_1 \cdot L^{VF}(\theta) - c2 \cdot \mathcal{H}(\pi(A_{t,k}|S_{t,k}, \theta)) \tag{6}$$

where c_1 is the weight of the critic loss, c_2 is a fixed entropy bonus defined in the original paper [4] and $\mathcal{H}$ is the entropy, defined in equation 7. The entropy of a policy in reinforcement learning is lower when the policy is more certain about which action to take, indicating a higher probability being assigned to one or a few actions over others. By maximizing this entropy, the agent is encouraged to maintain a higher level of randomness in its action selection, which promotes exploration.

$$\mathcal{H}(\pi(a|S_t, \theta)) = -\sum_t \pi(a|S_t, \theta) \log \pi(a|S_t, \theta) \tag{7}$$

Given these formulas, the weights update is computed as:

$$\theta_{t+1} = \theta_t + \alpha\nabla_\theta L^{total}(\theta) \tag{8}$$

2.3 Design Choices and Motivations

As previously mentioned, many parameters and hyperparameters were adopted directly from OpenAI's original work [4]. However, during the experiments, two parameters proved unsuitable for the developed algorithm.

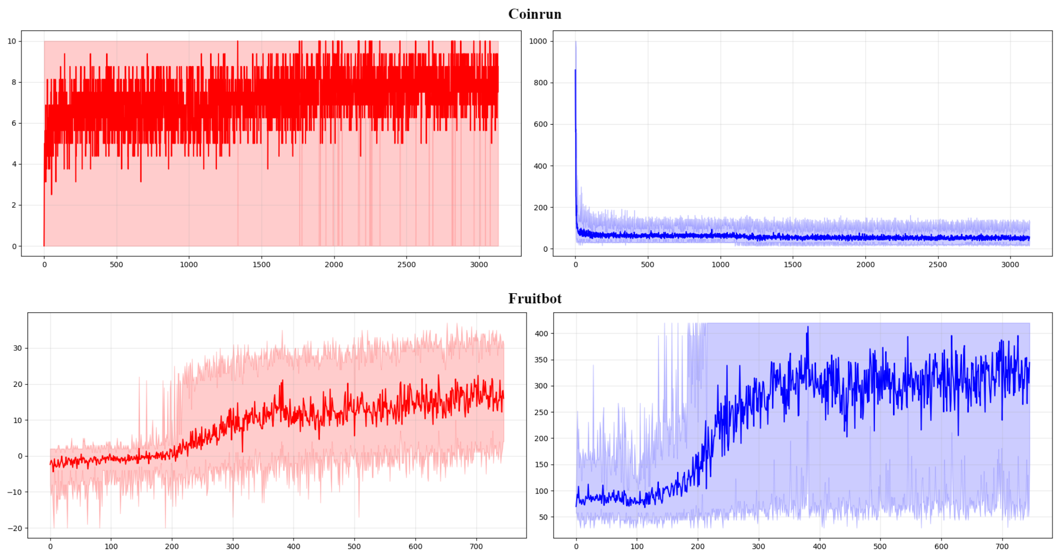

Figure 2: Training plots. The red line represents the average sum of rewards per episode across different environments. The blue line shows the average episode length across environments.

The first parameter that differs from the OpenAI formulation is the number of environments, which in turn affects the batch and minibatch size. OpenAI employed 64 parallel environments, but due to computational constraints, this number was reduced to 16. This adjustment aims to balance the benefits of training with more environments, which generally leads to faster and more stable training [1], and the need to maintain an appropriate and supported minibatch size for effective training.

The second parameter that differs from the original work is the learning rate, which was adapted to different games after several tests. The same configuration of PPO with a single learning rate varied the agent's behavior significantly during training. Specifically, it was observed that low learning rates led to little to no improvements, while high learning rates initially improved the sum of rewards within episodes but caused the phenomenon of catastrophic forgetting at a certain point during training.

Additionally, the original *procgen* paper did not specify certain implementation details, such as the precise version of Proximal Policy Optimization (PPO) used, the method for computing advantages, and whether to employ an action-value or a state-value function estimator. This omission necessitated additional research to effectively implement these critical components.

In addressing these gaps, the approach adopted here utilizes PPO as detailed in its original formulation [7]. It employs a simplified version of Generalized Advantage Estimation (GAE) that confines its calculations to the current batch of data, as outlined in 3. This approach mandates the use of a state-value estimator over an action-value estimator. This aspect simplifies the computational effort, decreases both the complexity and variability of the estimates, and enables a smooth transition between estimators with higher and lower biases through the adjustment of the λ parameter [8].

3 Experimental setup and results

During the training phase the agent's improvements were evaluated by tracking the sum of rewards within episodes, averaged across the different environments. Specifically, during the training phase, the Gym environment was configured to the easy setting, with backgrounds enabled and 200 different levels. To ensure reproducibility, a fixed seed was set for the environment and all external libraries. The agents demonstrated significant potential for improvement during the 3 million timesteps of training. Although this duration was limited due to the extensive time required, further improvements are expected, as the "standard" training time suggested by Procgen for the easy configuration of the environments is 25 million timesteps.

The tested games are:

- *Coinrun*: A simple platformer where the goal is to collect the coin at the far right of the level, with the player starting on the far left. The player must dodge stationary saw obstacles, enemies that pace back and forth, and chasms that lead to death. The best performance is achieved when the agent obtains a high reward in a short time. The maximum sum of rewards within an episode is 10.
- *Fruitbot*: A scrolling game where a robot must navigate between gaps in walls and collect fruit along the way. The player receives a positive reward for collecting a piece of fruit and a larger negative reward for mistakenly collecting a non-fruit object. At the end of the level, a big reward is given. The best performance is achieved when the agent obtains a high reward over a longer time. The theoretical bound representing the maximum sum of rewards within an episode is 32.4.

The training plots shown in Figure 2, demonstrate how both agents are able to learn according to the specific requirements of the games, successfully overcoming the diverse challenges presented. Additionally, the learning curves are very similar to those presented by OpenAI for the easy setting of the environments [4].

The abilities of these agents are also tested on an infinite set of unseen levels for 100,000 timesteps. The performance observed during this phase is compared to that of a random agent, as shown in Table 1.

	FruitBot		Coinrun	
	Avg reward	Avg Episode Length	Avg Reward	Avg Episode Length
Random Agent	-2.54	82.33	2.44	638.69
Trained Agent	16.15	320.68	6.83	55.18

Table 1: Average sum of all rewards and episode lengths obtained in the test environments.

The agents, on the test set, demonstrate results very close to the performance obtained during training, showing a strong ability to generalize to new, unseen scenarios.

Fruitbot learns very well; the only significant difficulties arise when it encounters a gate "blocked" by junk food or when there are both fruits and junk food near a gate. In these situations, the agent often crashes. In all other cases, it is generally able to reach the end of the level.

Coinrun seems to learn a very simple strategy, where it is sufficient to jump and move right to reach the end of the level. Sometimes these jumps appear calculated, avoiding obstacles that would end the level. Other times, the jumps seem almost automatic and careless of the landing consequences. It is possible that with further training, this technique could be refined.

4 Conclusion

This report explored the implementation of a Proximal Policy Optimization (PPO) approach to solve a subset of tasks in the Procgen Benchmark. The developed agents demonstrated a strong ability to learn and generalize, showing performance closely aligned with the results presented by OpenAI. The modifications made to the original algorithm, such as reducing the number of environments and adjusting the learning rate, proved effective in overcoming computational constraints and improving training stability.

The agents' performance in *fruitbot* and *coinrun* showcased their ability to adapt to specific game requirements. *Fruitbot* posed challenges mainly around gates blocked by junk food, while *coinrun* highlighted a simple yet effective strategy of jumping and moving right. Further training could potentially refine these strategies and improve overall performance.

Overall, the experiments validate the efficacy of PPO in high-dimensional, procedurally generated environments.

References

[1] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020. URL `https://arxiv.org/abs/2006.05990`.

[2] Yurou Chen, Fengyi Zhang, and Zhiyong Liu. Adaptive advantage estimation for actor-critic algorithms. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9534005.

[3] Yurou Chen, Fengyi Zhang, and Zhiyong Liu. Adaptive advantage estimation for actor-critic algorithms. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9534005.

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

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

[6] OpenAI. Procgen benchmark, 2023. URL `https://openai.com/index/procgen-benchmark/`. Accessed June 3, 2024.

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

[8] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.