Proximal Policy Optimization on ProcGen

Umberto Carlucci

Abstract

Introduction

ProcGen is a benchmark developed by OpenAI and consists of 16 unique environments created to test and evaluate generalization. In this report, a subset of the games provided by ProcGen will be addressed using Proximal Policy Optimization [1]. This family of actor-critic based, online policy gradient methods has been chosen for a wide set of reasons: the remarkable performances of PPO algorithms over benchmarks similar to ProcGen both in variety and complexity of the environments (e.g Atari Games); its relative ease of implementation with respect to other algorithms; its stability and robustness to changes in hyperparameters. The results provided show how these algorithms can perform well even with a limited amount of resources available.

Methodologies

Environment Setting

ProcGen is a very difficult benchmark and to reliably evaluate generalization performance of a PPO agent over it without a very high number of training hours or specific hardware, a subset of 2 games, with their gym environments implementation, was chosen:

- CoinRun: minigame in which the player has to reach a coin all the way to the right of the scenario; each level can contain traps, obstacles, enemies and boxes. This environment can be really challenging because the reward is either 0 (if the player takes too much time to complete the level or goes on a trap or an enemy) or 10 (the player touches the coin), there are no intermediate rewards on which the policy could calibrate.
- BigFish: in this minigame, the player is a small fish which has to become larger eating smaller fish while avoiding bigger ones. Fish can come in different size, colors and can spawn in both sides of the screen.

For each game, 2 environments were created:

- the first one to train the agent, instantiated with 200 levels to provide a good level of generalization and the easy-mode for what concerns the distribution of levels.
- the second one to test the agent, instantiated with an unlimited amount of levels, the easy-mode as distribution and a different seed, to assess the generalization capabilities acquired by the agent. The trained agent is compared to a randomly-acting one on this same testing environment.

Moreover, for each game a test with

```
use_background = True
```

is performed. With this flag on, the levels will now be generated with coloured backgrounds that should introduce more variability inside the environment.

Algorithm & Hyperparameters

PPO is a reinforcement learning algorithm that uses a specific loss function to limit policy variation. It uses specific hyperparameters:

- Lambda λ is useful when computing the Generalized Advantage Estimation: it weighs the difference between future rewards and close rewards in terms of a trade-off between bias and variance.
- Gamma γ defines how "far" in the future the agent looks when computing the expected reward.
- Clip controls how much the new policy can be different from the old one.
- Critic discount is a variable that scales the critic's loss. By adjusting it, it is possible to modulate the influence of the critic's feedback on the overall learning process.

These hyperparameters can influence the outcome of the experiments. The PPO algorithm provided in the code uses:

- the class *PPO_Storage* as experience replay to make the training more robust;

- the function *advantage* which relies on γ and λ to compute the advantage estimates, namely how much better an actions is compared to the average action at each state. γ is the discount factor, and determines how much future rewards are valued compared to immediate ones; λ describes the decay of the weighting factor over a weighted average of n-step returns, balancing between bias and variance;
- the function *learning_step* within the agent to generate batches of experience, to calculate the advantage for each of those batches and to update the network. It uses the **clip** parameter to limit changes in the policy.

The starting hyperparameters for CoinRun are standard values:

Hyperparameter	Value
Learning Rate	5e-5
Episodes	400
Epochs	3
Batch Size	256
Gamma	0.99
Lambda	0.95
Clip	0.2
Rollout Size	1024
Critic Discount	0.5

To help the generalization process, batches are also shuffled.

Neural Network

Both the actor and the critic are based on a shared convolutional neural network, created using PyTorch; the two components only differ by the last layer, since the actor has a number of outputs which is equal to the number of the possible actions within the game while the critic only outputs the value. The neural network consists of 3 convolutional layers of increasing depth and kernel size and decreasing stride; each layer is followed by a ReLU activation function and a BatchNormalization layer. The last layer before the two classifiers is a flattening layer; before the actor classifier there is a SoftMax. Actor and critic also share an Adam optimizer.

Evaluation

For each model, there are 2 versions kept and tested:

- The first one is the model fully trained.
- The second one is the model capable of maintaining the highest average across a certain amount of episodes (usually, 10%) of the total episodes chosen for the experiment. This is done to assess the generalization capabilities of the model throughout the training, to avoid overfitting in the later portions of the training and to have an insight on the results of the agent even if the training suffered of catastrophic forgetting.

Results

CoinRun

First Experiment The first experiment is carried out with the standard hyperparameters, described in the previous section. With this setup, the agent encounters what resembles to be a catastrophic forgetting situation, as shown in figure 1.

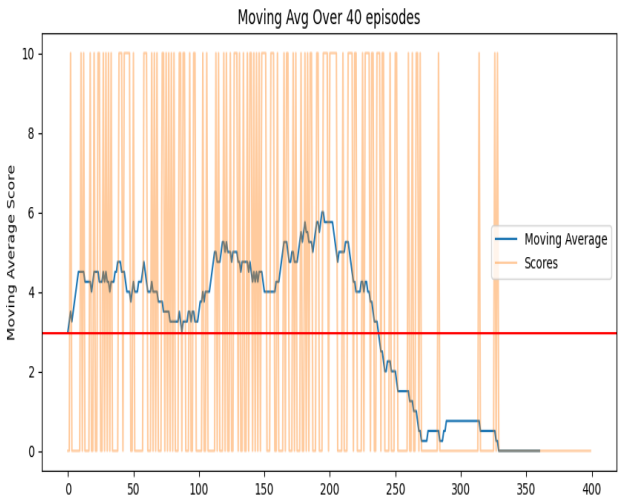

Figure 1: *The blue line is the moving average over 40 episodes, the yellow line represents the score for each episode, the horizontal red line is the global average*

This is maybe due to the fact that even with 200 levels the agent encounters a very diversified array of situations and "forgets" what it learned in the earlier phases. In fact, this statement is enforced by the results, as shown in Table 1. Between episode 150 and 220 the reward is increasing before suffering a total collapse around episode 240. The checkpoint is saved at episode 233 because in the previous 40 episodes the average reward was the highest within the whole experiment, with a peak of **6.0**. The number of timesteps in this experiment is 170k.

Agent	Env	Avg	Highest
Full	Training	2.95	10
Random	Testing	2.875	10
Full	Testing	0.0	0
Best (233)	Testing	5.375	10

Table 1: *The "checkpoint" policy demonstrates how the model was learning and capable of generalizing also to new levels, up to a certain point.*

Second Experiment With the second experiment, the clipping hyperparameter is instantiated to 0.25 instead of 0.2. This is done to encourage exploration while retaining a good equilibrium between exploration and old information in order to avoid the previous catastrophic forgetting situation. The graph in figure 2 shows how this effectively happens, even with smaller scores, as summarized in the table 2. The graph shows an increasing trend, the average during training is sensibly higher than the previous experiment but the highest average is a little bit lower, with a value of **5.75** at the end of episode 251. The fact that the highest average is computed in two episodes not far apart from each other (233 and 251), the growing trend registered towards the end and a training that at times seems stable (there is one major drop around episode 50 and 2 smaller ones around episode 150 and 250) might suggest that trying with more epochs could lead to better results and a more stable training.

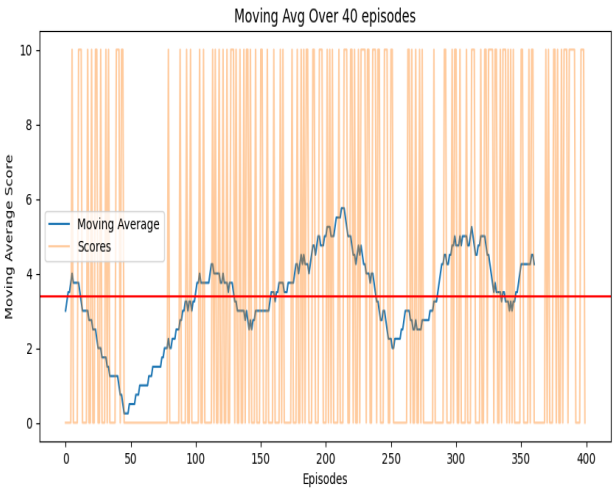

Figure 2

Agent	Env	Avg	Highest
Full	Training	3.4	10
Random	Testing	3	10
Full	Testing	3.875	10
Best (251)	Testing	3	10

Table 2

The fact that the best policy is almost 1 point off the fully trained one means that in the episodes between 252 and 400 the policy has been updated in a way that is beneficial for generalization.

Third Experiment As mentioned in the previous section, in this experiment the number of epochs is different, being doubled (from 3 to 6). This may help the agent to learn in a faster and more generalizing way, as graph 3 partially confirms.

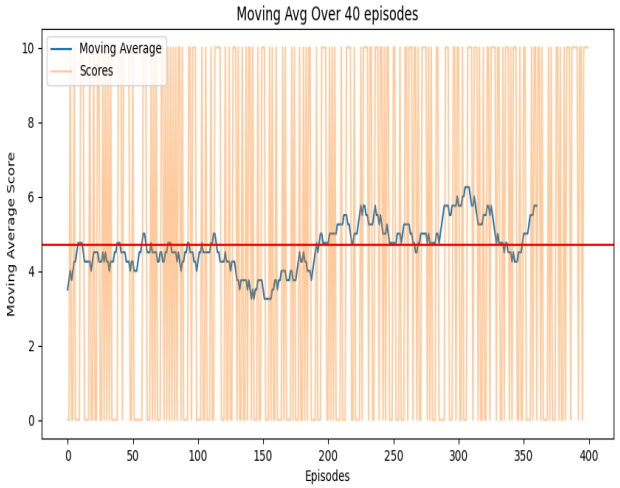

Figure 3

In this case, neither the catastrophic forgetting seen in the first experiment nor the decline at the beginning occurs, the global average is reasonably higher as shown both in the table 3 and the graph and the highest moving average is the better registered until now, with a peak of **6.25**, recorded at episode 344 (much later than the previous 2 experiments). Generally, the training seems more stable, with a minor drop around episode 150.

Agent	Env	Avg	Highest
Full	Training	4.7	10
Random	Testing	2.5	10
Full	Testing	5	10
Best (344)	Testing	5.125	10

Table 3

Fourth Experiment In this experiment gamma and lambda will be slightly reduced to check if and how performance is affected. The full set of hyperparameters is:

Hyperparameter	Value
Learning Rate	5e-5
Episodes	400
Epochs	6
Batch Size	256
Gamma	0.6
Lambda	0.6
Clip	0.25
Rollout Size	1024
Critic Discount	0.5

Agent	Env	Avg	Highest
Full	Training	3.875	10
Random	Testing	3.25	10
Full	Testing	4.625	10
Best (344)	Testing	4.5	10

Table 4

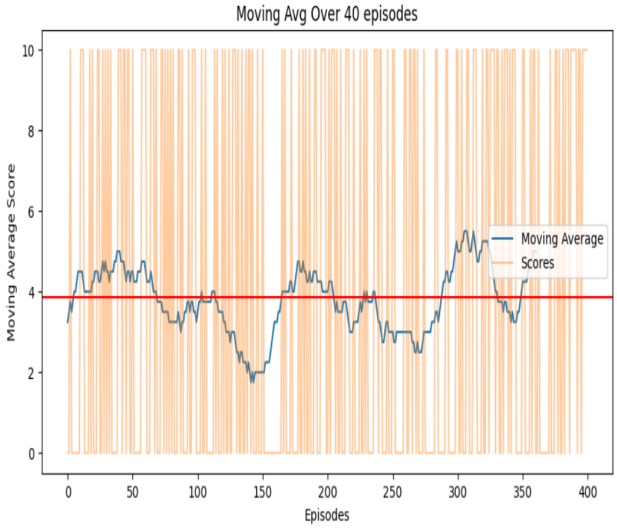

Figure 4

The agent has shown discrete generalization capabilities and a reasonably stable training. The highest average across a 40-episode span is **5.5**, at level 344 and there are no huge differences between the fully trained model and the best scoring one. With more experimentation on these hyperparameters, it is very likely that the agent will become capable of achieving higher rewards.

Fifth Experiment As last experiment on CoinRun, the best parameters found until now will be used to train and test a model on environments with changing backgrounds.

Agent	Env	Avg	Highest
Full	Training	2.10	10
Random	Testing	2.625	10
Full	Testing	0	0
Best (77)	Testing	4.375	10

Table 5

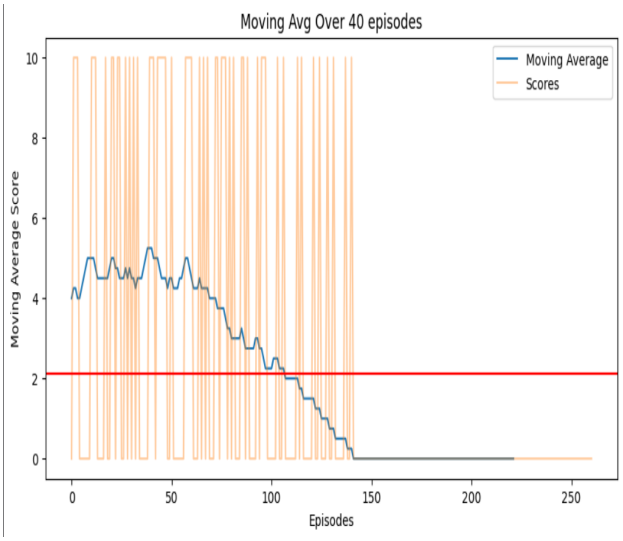

Figure 5

The agent shows a clear situation of catastrophic forgetting. Like in the first experiment, this is happening because with this set of hyperparameters the agent is not capable of learning fast enough before encountering levels too different from each other. This can be confirmed also by the inconsistent peaks at the start. However, the model saved at level 77 with an average of **5.25** scores an average of 4.375 on the testing environment: this good score achieved by an agent stopped at the earlier phases of the training is surely due to the fact that the agent understands early enough, also with the backgrounds on, that going to the right may result in a high reward. The catastrophic forgetting occurs earlier than the First Experiment probably because episodes are much more diverse.

BigFish

Since the 6 epochs experiment carried out on CoinRun gave good results with an overall reasonable amount of training time, it will be kept as hyperparameter.

First Experiment In the table below the results of the first experiment on BigFish are summarized.

Agent	Env	Avg	Highest
Full	Training	0.96	40
Random	Testing	0.82	6
Full	Testing	1.18	7
Best	Testing	1.0375	9

Table 6

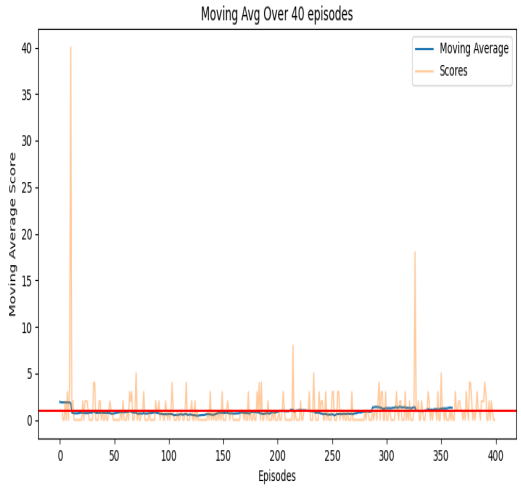

Figure 6

Since there is almost no visible sign of consistent learning and no remarkable highscores, the number of episodes and the learning rate will be, respectively, raised to 1600 and 1e-4.

Second Experiment The agent doesn't achieve very high scores on the training but unexpectedly the best model generalizes relatively well during the test phase. The best model is saved at episode n. 1303 with a peak in the average scores of **1.8**, very later in the training, supporting the slow learning hypothesis over the BigFish environment. The best model achieves also the highest between the highscores.

Agent	Env	Avg	Highest
Full	Training	1.09	27
Random	Testing	0.84	10
Full	Testing	0.84	12
Best (1303)	Testing	3.14	40

Table 7

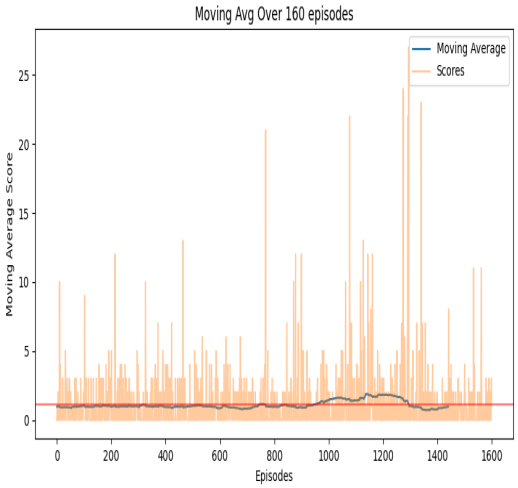

Figure 7

The graph seems more promising than the one before, with the higher scores occurring in the later parts of the training with a certain periodicity. The huge drop in performance is maybe due to the fact that the agent is in an exploration phase.

Third Experiment Since the last model was saved at episode 1303 and then registered a performance drop, the number of episodes has been slightly reduced, to 1200, while experimenting changes on other hyperparameters, bringing gamma and lambda respectively to 0.9 and 0.6, making the agent more focused on more immediate rewards.

Agent	Env	Avg	Highest
Full	Training	0.99	40
Random	Testing	0.95	8
Full	Testing	0.78	13
Best (730)	Testing	0.93	7

Table 8

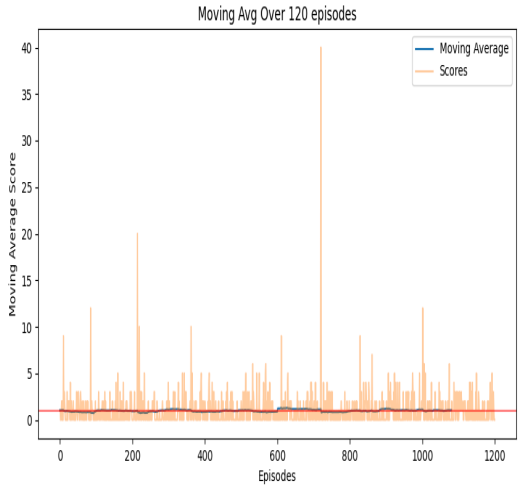

Figure 8

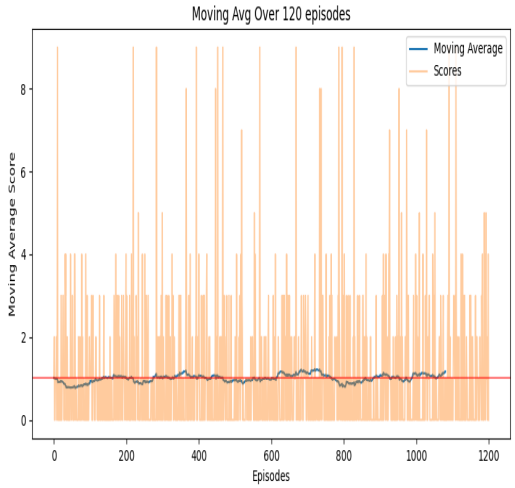

Figure 9

The agent is now too focused on immediate rewards, since the graph is almost flat near the global average and there is only one remarkable highscore.

Fourth Experiment Since experimenting over gamma and lambda could be a promising way but at the same time a very time consuming one and given both the very slow convergence and learning over BigFish, the last experiment will use a higher learning rate (4e-4) and different critic discount (0.75), to check whether those changing those hyperparameters could give a different outcome.

Agent	Env	Avg	Highest
Full	Training	1.01	9
Random	Testing	0.875	7
Full	Testing	0.95	15
Best (838)	Testing	0.9	9

Table 9

This experiment offers very interesting results in terms of consistency. Table 9 shows a training average very similar to the one obtained in the other experiments but with a much lower highscore: this implies a more compact range of scores and more consistency and that situation could be interpreted as the agent effectively being capable of learning steadily and consistently. Graph 9 confirms this statement, since it displays the most consistent results throughout all the experiment carried out on BigFish. Also the fact that the highest moving average (**1.2250**) is stored at episode 838, after over than 2/3 of the training, is a further confirmation.

Fifth Experiment As before, the last experiment uses the best performing hyperparameters (Second Experiment) to train and test the agent over environments with changing backgrounds. The highest moving average is unexpectedly not that far from the one achieved in the no-background experiment, with **1.65** at episode 731. The model doesn't show huge learning capabilities during the training but the best scoring model generalizes well, losing only 0.3 points with respect to the agent trained in the Second Experiment. It's clear that some hyperparameter tuning could be beneficial, since the best scoring model has been discovered in the first half of the training. However, the average between this experiment and the one with the same hyperparameters is exactly the same, underlining the adaptability of this agent over changing and more diverse environments.

Agent	Env	Avg	Highest
Full	Training	1.09	40
Random	Testing	0.865	9
Full	Testing	0.88	10
Best (731)	Testing	2.85	40

Table 10

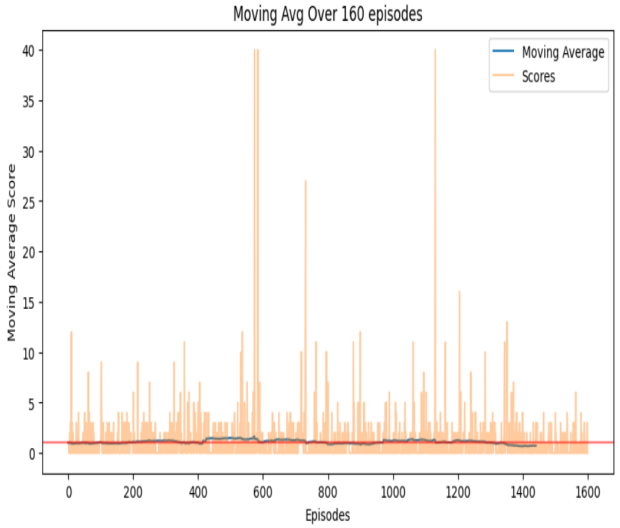

Figure 10

Discussion

The PPO agent seemed capable of generalizing across the two different games. The training on both games displays promising results, suggesting that more timesteps could be beneficial, especially for BigFish, given the slow but steady improvements it has shown during training. For example, the fourth experiment on BigFish seems to be the most interesting, as the agent shows the ability to balance exploration and exploitation very well: it has a very small range for scoring (0 to 8), but shows a similar average to more successful experiments (with wider ranges for scores), which means that it must have developed some consistency in the process. Further tests with this agent could include training for a larger amount of timesteps (the developers suggest at least 25M timesteps [2], while the experiments carried out in this project range from 30k to 260k due to computational resources) but also on more levels, since the authors suggest that an agent needs to be trained on 500-1000 levels[1] to generalize on unseen ones. The agent has shown remarkable robustness to change in the hyperparameters, providing good results also with different values of λ and γ, especially on ProcGen. On both games, the robustness of the agent has been proven trying to use the hyperparameters that worked well on the no-background environment and showing how the best scoring agent, in both cases, was able to achieve good results also with the backgrounds on. Even if the fully trained models over BigFish were underperforming, the ones with the highest moving average were performing well, both with and without the backgrounds.

[1] https://openai.com/index/procgen-benchmark/

Limitations and Conclusions

As anticipated in the previous section, the main limit of this project was the huge amount of resources required by ProcGen to achieve good results. CoinRun achieved good results probably because it is the easiest one between the 16 environments while it was not possible to show the same performance (on the fully trained agents) on BigFish due to the aforementioned computational constraints. However, even without these resources, it was possible to obtain good scores in almost all the experiments on CoinRun, while it was possible to overcome the random agent in various degrees throughout the experiments on BigFish, uncovering the probable influence that different hyperparameters may have during training. Also, the PPO agent provided for this project has been tried on other ProcGen games but the training times were prohibitive or the results without the very large amount of timesteps required were unsatisfactory. Another possible enrichment of this study could involve the use of a larger neural network: many of the initial tests were executed with two different optimizers and also two different networks, but the training was extremely slow. Despite the challenges and the limitations encountered, this project has showcased the potential of the PPO agent in tackling tasks like ProcGen.

References

[1] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[2] Karl Cobbe et al. *Leveraging Procedural Generation to Benchmark Reinforcement Learning*. 2020. arXiv: 1912.01588 [cs.LG]. URL: https://arxiv.org/abs/1912.01588.