Procgen-Solving agent using PPO and Distillation network strategy

Federico Rullo
University of Bologna
federico.rullo@studio.unibo.it

Abstract

This paper presents a Reinforcement Learning agent designed to solve a subset of OpenAI's Procgen Benchmark environments using Proximal Policy Optimization integrating an exploration technique called Network Distillation. The hypothesis is that this technique will allow accelerated learning reducing the amount of epochs needed for the agent to reach an acceptable result, while also improving generalization so that the same configuration of the agent can be applied to learn the entire set of Procgen's generated environments.

1 Introduction

OpenAI's Procgen Benchmark presented in [1] is a collection of 16 procedurally generated environments designed to assess generalization in Reinforcement Learning agents. Standard Reinforcement Learning algorithms often struggle in these environments due to the vast variability across states, this is because training is not done on a single level or a single instance of the environment, like in the Atari Environments where only one level is selected and we try to maximize the reward for that given level, here multiple configurations are presented where no two instances of the environment, called levels, are the same. This allows us to assess the generalization capability of an agent but also exponentially increases the complexity of these environments. Proximal Policy Optimization has been a popular choice in this context due to its balance between exploration and exploitation, but improvement in learning speed and generalization remains challenging. In this work, I explore whether incorporating an exploration technique called Network Distillation can enhance the performance of PPO-based agents. Network Distillation is a technique involving training a smaller model, the student, to mimic the behaviour of a larger model, the teacher, which is trained on the actual environment itself, the goal is to compress knowledge and improve generalization. To conduct this experiment a subset of three environments have been selected from the Procgen list:

1. Heist
2. Bossfight
3. Leaper

These environments should provide a good benchmark for complexity.

2 Related Work

2.1 Proximal Policy Optimization

Proximal Policy Optimization is an algorithm developed by [4] designed to efficiently optimize policy and improve generalization. Proximal Policy Optimization is an on-policy method which means that it directly optimizes a policy using an optimization algorithm such as Gradient Descent to maximise the expected cumulative reward, a distinctive feature of PPO its clipped objective function, this

ensures the new policy does not deviate too drastically from the previous policy, this is because if an update drastically changes the policy yielding worst results it is impossible to go back to the previous state. To achieve this, PPO introduces a clipping term in the objective function which penalizes large changes in the policy ratio. Another feature PPO implements is an Advantage function, which guides policy updates by measuring how much better an action is compared to the average action taken in a given state. These features allow us to improve our policy effectively, and we end up with an update like this:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{1}$$

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

Where π_θ is the parameterized policy by θ, $\hat{A}_t$ is the advantage estimate at time step t, and ϵ is a constant controlling the clip range. PPO updates the policy by maximising the clipped objective function using Gradient Ascent this is also called the Critic Value, which helps reduce the variance of the policy gradient estimates and is trained using Mean Squared Error Loss.

2.2 Network Distillation in RL

Network Distillation or Policy Distillation is a technique introduced in [3] where knowledge from one or multiple trained networks is transferred into a simpler, more efficient network. Distillation can be used to improve the efficiency and performance of PPO in scenarios where the original Policy Network struggles to generalize or is too large or complex for real-time usage. Policy Distillation works by having a teacher network, often large and computationally expensive, trained on the environment, the results of this network are then used to train a student network which is typically smaller and faster. The goal of this technique is to maintain the performance of large models while reducing the computational footprint. In Proximal Policy Optimization distillation is applied as follows:

1. Train the teacher network on the environment
2. After training, the teacher network is distilled into a smaller network, called the student, by training it to imitate the behaviour of the teacher.
3. In alternative, distillation may be used during training when multiple teachers are distilled into a single policy, this enables the student to learn multi-tasking.

PPO distillation follows a supervised learning approach where the student is trained to minimize the discrepancy between its output and the output of the teacher network. Specifically, the student attempts to match the probability distribution over actions produced by the teacher network, often using the Kullnack-Leibler divergence as the loss function:

$$L_{distill}(\theta) = D_{KL}(\pi_{teacher}(a|s)||\pi_{student}(a|s;\theta)) \tag{3}$$

where:

- $\pi_{teacher}(a|s)$ is the teacher's probability distribution over actions a given state s.
- $\pi_{student}(a|s;\theta)$ is the student's policy parameterized by θ-
- D_{KL} is the KL divergence

2.3 IMPALA Block

The Impala Block is a component introduced by DeepMind for the IMPALA Framework in [2]. It is used for feature extraction in Deep Reinforcement Learning which involves complex visual inputs. The Impala Block is composed of a Convolutional Layer, MaxPooled for downsampling feature maps, and two Residual BLocks these blocks facilitate training avoiding issues such as vanishing gradients. Because Procgen Environments are procedurally generated they tend to be rather high in complexity, hence the integration of this block in the network facilitates extracting features from frames.

3 Methodology

This section will describe the Agent and training configuration, as well as the choices made for the Hyperpamaeters.

learning_rate	$5e-4$
distillation_learning_rate	$2e-4$
γ	0.999
β	0.01
λ	0.95
ϵ	0.2
val_coef	0.5
num_environments	24
num_epochs	2M
num_trajectory_steps	256
training_ppo_epochs	2
batch_epochs	8
mini_batch_per_epochs	2048
distillation_scale	100

Table 1: Caption

3.1 Hyperparameters

The choice of hyperparameters is illustrated in the hyperparameters table1. This configuration follows the main procgen paper with slight modifications. The introduction of a distillation scale which controls the impact of the distillation loss relative to other loss components and ensures effective knowledge transfer without dominating the training process has been selected by testing different values, starting with a value of 50 it was observed that doubling that value would yield better results. Another modification is in the number of environments created for training, this parameter controls how many environments are created when we instantiate the environment, and this has a high impact on the speed of training but it is memory heavy, the original procgen paper instantiates 64 agents but because this configuration uses three neural networks instead of just one, and because the resources used by openAi were not available, after some trials, to see which number did not saturate the GPU memory, the number of instantiated environments, which left more memory available to increase the mini_batch_size per training epochs. This allowed us to still retain the speed of having multiple environments running at once while also making the agent able to experience more data during training.

3.2 Agent

The agent is composed of four main components:

- The main Network is composed of the actor and the critics for both intrinsic and extrinsic values
- The teacher network which learns extrinsic values for the observation
- The student network which is used to calculate the loss of the distillation components
- The Trajectory memory is used to store and sample the experience used to train the agent and also calculates the advantages and intrinsic and extrinsic value references.

To optimize the Main network and the teacher network the Adam optimizer.

3.2.1 Main Network Architecture

Initially based on the architecture from the *A2C* model, which had 3 convolutional layers shared for both actor and critic, both using linear layers to return the logits and the values, the convolutional part of the network was later modified because this simple configuration was not appropriate to the complexity of the Procgen environments. Hence, instead of simple convolutional layers Impala Blocks from [2] were integrated, allowing the network to properly extract features from the complex observations, these blocks were also extended with a Dropout layer to avoid overfitting. As stated before two critic networks are used instead of only one because we have to return two kinds of values, the extrinsic values, which are from the environment itself and the intrinsic values, which are used to perform distillation on the network. Both Actors and critics share the same configuration: Two

Linear layers followed by a ReLU activation function, with the only difference being that the intrinsic critic has only one layer since it will interact with the teacher and student networks.

3.2.2 Distilled Networks

Two distilled networks are instantiated for this agent, a teacher network and a student network, where the student is not trained but is used to calculate the distillation loss for the teacher. Both networks share the same configuration:

- Two ImpalaBlocks
- DropoutLayer
- Two Linear Layers
- ReLU activation layers

This network serves as a distilled reference model for the main network and is trained to approximate the intrinsic value functions more precisely. Integration of Distillation is not done by exactly following the main paper [3]. In the main paper, two heads are trained while here, because distillation was integrated after having the Base PPO agent, the results of distillation are passed to the main network through a gym wrapper which returns the actual reward from the environment (extrinsic reward) and the reward, or values, from the distilled networks (intrinsic), the latter reward serves as a target for the main network. The loss of the distillation is computed separately from the main loss, the Mean Squared Error is taken between the student and the teacher.

$$LOSS_{distill} = MSE(ReferenceNET(s), TrainedNET(s)) \tag{4}$$

3.3 Memory Handling

Experience is handled by a class called *DistilledTrajectories* responsible for storing and managing different trajectories experienced by the agent through the interaction with the environment. These trajectories are:

- **Observations**
- **Actions**
- **Rewards**
- **Rewards Sum**
- **Actions Log Probabilities**
- **Extrinsic Values**
- **Intrinsic Values**
- **Advantages**
- **Extrinsic Reference Values**
- **Intrinsic Reference Values**

These trajectories are instantiated as Tensors, and at each iteration, a copy of them is stored, once iteration is done populating the trajectories a last observation and values are stored.

3.3.1 Computing Advantages

The Memory class is responsible for calculating the advantages, since the various trajectories are already stored here it was easier to make this class handle them as well. This function also returns the discounted returns for both intrinsic and extrinsic rewards. To compute the advantages the Generalized Advantage Estimation method is used, this helps in reducing variance when estimating advantages by incorporating tradeoffs between the bias and the variance. To compute the advantages loop through the trajectory in reverse, doing a backward pass make sure the future discounted rewards and advantages are calculated. Computation of advantages is done by first calculating the Temporal-Difference Error:

$$\delta = R_t + \gamma * v_{\pi_{t+1}}(s) * (1 - done) - v_{\pi_t}(s) \tag{5}$$

Environment	PPO (Distilled)
Heist	3.7
Bossfight	0.74
Leaper	2.82

Table 2: Highest mean Reward Achieved by Distilled Agent and Random Agent on testing of 200k steps

Where γ is the discount factor for future rewards, $done$ is a flag indicating whether that episode has finished, so no further rewards are considered. Because of distillation, the values are calculated by summing the extrinsic and intrinsic values:

$$v_{\pi_t} = Vext_{\pi_t} + Vint_{\pi_t} \tag{6}$$

$$v_{\pi_{t+1}} = Vext_{\pi_{t+1}} + Vext_{\pi_{t+1}} \tag{7}$$

The advantages are computed recursively by using GAE:

$$\hat{A}_t = \delta + \gamma * \lambda * (1 - done) * \hat{A}_{t+1} \tag{8}$$

Where $lambda$ is a weighting parameter used in GAE to control the tradeoff between bias and variance, this allows propagation advantages backwards through time. Then Returns are computed as:

$$G_t = \hat{A}_t + (Vext_{\pi_t} + Vint_{\pi_t}) \tag{9}$$

Finally, the advantages are normalized by:

$$\hat{A}_t = \frac{(\hat{A}_t - mean(\hat{A}_t))}{std(\hat{A}_t) + 1e - 8} \tag{10}$$

Where $1e - 8$ is a constant to prevent division by zero.

3.3.2 Computing Intrinsic and Extrinsic Reference Values

This function follows almost the same procedure as for computing advantages, computing reference values for both intrinsic and extrinsic rewards, which allows balancing learning and improving exploration and exploitation, facilitating learning in complex environments.

4 Results

The agent was trained for 2 million epochs to evaluate the effectiveness of the distillation technique compared to the baseline training of 8 million epochs used in the original Procgen paper. The hypothesis was that distillation could reduce the number of epochs required to achieve acceptable performance. The agent was trained on an easier distribution of 200 levels. On average, training took approximately five hours per environment, depending on its complexity. The agent's performance varied significantly, achieving rewards either marginally better than a random agent or performing poorly. As shown in table 2 Although network distillation allowed the agent to outperform a random agent with a smaller set of epochs, it did not solve all environments. For example, the "Bossfight" environment proved too challenging for the agent in the limited amount of training time it had, as the agent only has one life per episode, ending the level after being hit. The other two tested environments, "Heist" and "Leaper" allowed for more agent exploration, but slowed down learning sensibly. Let's then analyze the worst-performing environment, "boss fight", by looking at the graphs in figure 1 at the mean rewards in a graph we can see that there are occasional spikes in reward values but they are rather close to zero, indicating limited success in this environment, this is probably because the agent doesn't have time to explore before the end of the episode. This kind of behaviour can also be observed in the action and distillation loss, regarding the action loss, fluctuates around the same values this indicates that the agent is not making substantial changes to its actions and the policy is relatively stable throughout training. Another problem is that the distillation loss is still really high which means that the environment is still too complex for the agent after these many epochs.

Figure 1: Bossfight training

5 Conclusion and Future Work

This study developed a PPO-based agent employing network distillation to tackle a subset of OpenAI's Procgen benchmark. While training for a reduced number of epochs produced promising results, the agent did not reach state-of-the-art performance levels and did not manage to solve all environments. Nevertheless, distillation played a crucial role in accelerating training, allowing the agent to engage with the game after fewer runs compared to the baseline PPO model. Future work could focus on incorporating hyperparameter optimization and extending the training duration to a higher number of epochs, which may yield more competitive results.

References

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

[2] Lasse Espeholt et al. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *CoRR* abs/1802.01561 (2018). arXiv: 1802.01561. URL: http://arxiv.org/abs/1802.01561.

[3] Sam Green, Craig M. Vineyard, and Çetin Kaya Koç. "Distillation Strategies for Proximal Policy Optimization". In: *CoRR* abs/1901.08128 (2019). arXiv: 1901.08128. URL: http://arxiv.org/abs/1901.08128.

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