Crossing the Freeway with Proximal Policy Optimization and Random Network Distillation

Hiari Pizzini Cavagna
Autonomous and Adaptive Systems Course
University of Bologna, Department of Computer Science and Engineering
hiari.pizzinicavagna@studio.unibo.it

Abstract

We discuss the implementation of the PPO algorithm (Schulman et al., 2017b) with an Exploration Bonus using Random Network Distillation (Burda et al., 2018), in order to solve Atari's hard exploration games. We started with the implementation of a PPO agent to play the Atari game Freeway, then we extended the implementation with the RND Exploration Bonus comparing and observing the results on Freeway and Venture.

1 Introduction

Playing Atari games has been a classical challenge in Reinforcement Learning where, over the years, various algorithm and solutions has been proposed, achieving great performance across all kind of Atari games. In *Human-level control through deep reinforcement learning* (V., K., and et al., 2016) a Deep Q Network agent is implemented to solve many Atari games, achieving "superhuman" performances in numerous cases. However, there were certain games where the algorithm didn't managed to obtain any points. In fact, hard exploration games such as Venture, Private-Eye and the famous Montezuma's Revenge are difficult to play given their sparse reward matrix, demanding an high grade of exploration. One possible way to address these kind of games was given by *Exploration by Random Network Distillation* (RND) introduced by Burda, Edwards, Storkey, and Klimov (2018), which provided the state-of-the-art performance at the time on various challenging Atari exploration games.
In the following chapters, we will discuss the implementation of the PPO algorithm, focusing on playing the game Freeway as an example. Then, it will be discussed the implementation of the RND algorithm as a possible improvement to the previous implementation and to play Venture, an hard exploration game.

2 Proximal Policy Optimization Algorithm

2.1 The algorithm

The Proximal Policy Optimization Algorithm is a well-known Policy Gradient Method that has good performance playing Atari games, proposing a simple and general way to approximate the policy. PPO extends the solution proposed by the Trust Region Policy Optimization method (Schulman et al., 2017a) which maximize the following objective function:

$$\max_{\theta} \quad L^{CPI}(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right],$$

$$\text{subj. to} \quad \hat{E}_t \left[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)] \right] \leq \delta$$

Preprint. Under review.

The objective function is constrained because it may lead to destructive large policy updates (also possible to be implemented as a penalty with a coefficient β).
Therefore, the main idea of PPO is to maximize a clipped surrogate objective, equal to:

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

where

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

Where $\hat{A}_t$ is the estimated advantage, $\pi_\theta(a_t|s_t)$ represents the probability given by the policy of the action a in the state s at the time t and ϵ is the clipping hyperparameter. It can be noticed how the second term of the min function is a clipped version of the first one, in which $r(\theta)$ is bounded on both directions by $1 \pm \epsilon$. Then, the clipped $r(\theta)$ is multiplied for $\hat{A}_t$, and the minimum between the product and L^{CPI} is taken. As such, the final objective is a lower bound of L^{CPI}.

The advantages estimations, given the length of the trajectory T, are calculated as:

$$\hat{A}_t = \delta_t + (\epsilon\lambda)\delta_{t+1} + \ldots + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

Moreover, the algorithm minimizes the critic loss $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$.
Thus, the final loss is given by:

$$L^{CLIP+VF+S}(\theta_t) \ = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)].$$

where S represents an entropy bonus and $c1, c2$ are coefficients.

The algorithm uses N parallel actors, collecting T timesteps at every iteration, then the loss optimization is performed over K epochs.

Algorithm 1 PPO, Actor-Critic Style

for iteration=1,2… **do**
  for actor=1,2,…,N **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Calculate advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

2.2 Implementation

We implemented the PPO algorithm to create an agent capable of playing the Atari game Freeway, following the paper's hyperparameters for Atari games. The implementation involves 8 parallel actors playing for 128 timesteps, for a total of 10 million steps in the environment. The network architecture is the same used in the PPO paper, described in V., K., and et al. (2016), with one head for the actor and one for the critic.

The implementation is done using the Open AI Gym library, maintained by the Farama Foundation, and EnvPool. The Gym library provides a collection of environments, ranging from simple controls to Atari games, allowing easy algorithm testing through a simple API. Moreover, it offers various Environment Wrappers that are used in this implementation in order to resize, stack and skip frames as the original work suggest. EnvPool is a C++ library compatible with Open AI Gym APIs, which offers a batched environment pool with great performance, we used it to manage the 8 parallel actors.

It is important to note that different random seeds can have a significant influence on the results, as possible to see in the comparison between seeds in the PPO paper, thus we will conduct the experiments using 3 different seeds. Moreover, the total number of iteration steps are consistent with the original work, where 10 thousand steps are equivalent to 40 million frames. This calculation takes into account the 4 stacked frames for 128 timesteps in the environment for each PPO step, performed by 8 parallel agents.

We choose to use the game Freeway as example: a simple Atari game where the player assumes the role of a chicken attempting to cross a street where cars are passing. Each time the player gets hit by a car, it falls back. Crossing the street gives one point, and there is a time limit.

2.3 Results

We tested the implementation using 3 seeds, obtaining different outcomes. With the best seed, as possible to see in Figure 1, after a brief period the network starts to gain more and more rewards, reaching an impressive score of 30 after only 2000 steps. Then, the performance slightly improves eventually reaching an impressive 34 [1] as the final mean score, obtaining a state of the art performance. This result is consistent and slightly better wrt the original work where it's reported 32 as final mean reward. On the other hand, the others two seeds once reaching a score of 23 stop to improve till the end of the training. In the Git-hub repository is possible to find a video of the best agent playing.

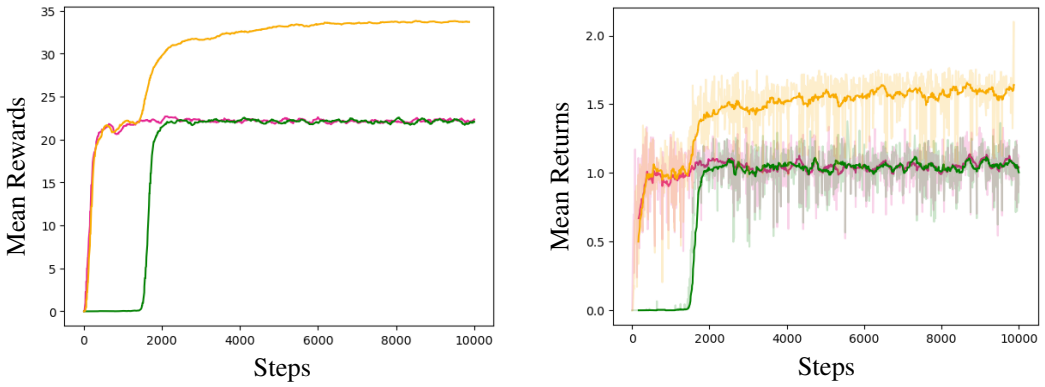

Figure 1: Results of the Freeway PPO implementation using the seeds 3047 (yellow), 1910 (pink) and 42 (green). Each step consists of 128 timesteps in the environment, performed by 8 parallel agents, for a total of 40 million frames. The mean reward is calculated among the last 100 concluded episodes.

3 Random Network Distillation

The work *"Exploration by Random Network Distillation"* (Burda et al., 2018) introduced an exploration bonus using two additional networks in order to detect novelty in the states encountered by the agent. Thus, the exploration bonus is summed to the environment reward, producing a total reward at time t equal to $r_t = e_t + i_t$, where e_t is the reward given by the environment - the *extrinsic* reward - and i_t is the exploration bonus - the *intrinsic* reward.

3.1 The intrinsic reward

The generation of the intrinsic reward involves two neural networks: the *target* and the *predictor* and it is based on the distillation process, which involves distilling a random neural network into a trained one. The target network is randomly initialized and sets the so called prediction problem: given the same input to both the networks, the predictor must "guess" the output of the target network.
More precisely, given the observation produced by the target network $f : \mathcal{O} \rightarrow \mathcal{R}^k$ and the one produced by the predictor $\hat{f} : \mathcal{O} \rightarrow \mathcal{R}^k$, the predictor is trained by minimizing the loss given by the

[1] To quantify the score, is possible to see the benchmark on Benchmark of Atari 2600 Freeway

mean squared error (MSE) between f and $\hat{f}$. In this way states dissimilar from the previous ones - produced by exploration - will result in a higher MSE, detecting novelty. The original work doesn't provide details about the output of the two networks, so we used the dimension of the action's space, as it is for the actor, and the calculation of the reward will follow the example given in a article on Data Iku[2]. Thus, the intrinsic reward i at a certain time t is equal to the normalized MSE using the Euclidean Norm, based on the observation of the state s_{t+1}:

$$i_t = ||\hat{f}(s_{t+1}) - f(s_{t+1})||_2^2.$$

3.2 Episodic and non-episodic returns

The paper argues that the exploration works better if the intrinsic rewards are calculated as non-episodic, claiming that this is actually how humans explore games. In fact, it is common that exploration leads to dangerous situations that could result in a game over. For instance, consider a scenario of a player reaching a cliff where there may be the possibility of jumping to the other side, where there could be rewards. However, the jump is difficult, and failing it would end the game. If the returns are episodic, the agent would avoid this dangerous situation that could end the episode, on the contrary, a human would repeat the episode multiple times until reaching the other side, if possible. The non-episodic intrinsic returns R_I and the episodic extrinsic returns R_E are combined together as a sum, having $R = R_E + R_I$ and are used to fit two value heads (critics) V_E and V_I.

Algorithm 2 RND pseudo-code

$N \leftarrow$ number of rollouts
$N_{opt} \leftarrow$ number of optimization steps
$K \leftarrow$ length of rollout
$M \leftarrow$ number of initial steps for initializing observation normalization
$t = 0$
Sample State $s_0 \sim p_0(s_0)$
for $m = 1$ to M **do**
    Sample $a_t \sim$ Uniform(a_t)
    Sample $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    Update observation normalization parameters using s_{t+1}
    $t+ = 1$
end for
for $i = 1$ to N **do**
    for $j = 1$ to K **do**
        Sample $a_t \sim \pi(a_t|s_t)$
        Sample $s_{t+1}, e_t \sim p(s_{t+1}, e_t|s_t, a_t)$
        Calculate intrinsic reward $i_t = ||\hat{f}(s_{t+1}) - f(s_{t+1})||^2$
        Add $s_t, s_{t+1}, a_t, e_t, i_t$ to optimization batch B_i
        Update reward normalization parameters using i_t
        $t+ = 1$
    end for
    Normalize the intrinsic rewards contained in B_i
    Calculate returns $R_{I,i}$ and advantages $A_{I,i}$ for intrinsic reward
    Calculate returns $R_{E,i}$ and advantages $A_{E,i}$ for extrinsic reward
    Calculate combined advantages $A_i = A_{I,i} + A_{E,i}$
    Update observation normalization parameters using B_i
    for $j = 1$ to N_{opt} **do**
        Optimize θ_π with respect to PPO loss on batch B_i, R_i, A_i using Adam
        Optimize $\theta_{\hat{f}}$ with respect to distillation loss on B_i using Adam
    end for
end for

[2]The article "Random Network Distillation: A New Take on Curiosity-Driven Learning"

3.3 Implementation

The provided RND algorithm (Algorithm 2) extends the PPO algorithm (Algorithm 1), by adding the management of the intrinsic reward and the optimization of the RND networks. We implemented both algorithms together, offering a switch that allows to activate the RND exploration bonus or to use PPO. As possible to see in Algorithm 2 it is required to normalize both the state and the intrinsic reward, to maintain consistency across different environments. The normalized state is only used for the predictor and the target networks. In order to do that, we used the Gym's normalization class RunningStdMean[3], which implements the Welford algorithm to calculate the mean and standard deviation of a stream of elements. In the original work, the train is performed using 128 parallel agents for 30 thousand steps, resulting in a total of 2 billion frames, a lot more compared to the 40 million frames used in the PPO implementation. Due to the increased computational power and memory required wrt PPO, we conducted only a few experiments with this algorithm.
Initially, we tested the RND algorithm as a possible improvement of the PPO implementation of Freeway, using the same hyperparameters of the PPO implementation. Then, we experimented the algorithm to play the hard exploration game Venture, also used as an example in the original work. As the paper says, we used the same network architectures as those used in the PPO implementation, for all the networks. The original work also provides comparison with a RNN architecture, obtaining similar results. We also used the same hyperparameters as those reported in the paper, however, due to hardware and time constraints, we used only 32 parallel environment instead of 128 and we trained for only 8 thousand steps instead of 30 thousand, for a total of 131 million frames.

3.4 Results

3.4.1 Experiment: Freeway with RND bonus

In Freeway the actual reward is given only when successfully crossing the street for the first time, and it's initially obtain by PPO only by chance. Moreover, as shown in Figure 1, only one seed is able to achieve a final score of 34, while the other two remain constant, unable to improve. We hypothesized that the addition the exploration bonus could lead to a faster train in terms of steps by rewarding the "exploration" of the street. For this experiment we implemented the RND algorithm, while keeping the same hyperparameters and number of actors of the PPO implementation. In Figure 2 we show a comparison between the results. In this setup, all the seeds obtained a positive score in the early stages. However, the final outcome did not show a general improvement. In fact, only one seed improved, while seed 42 seemed to perform worse, with a decline in its initial score. We think this could be due to the agent's inclination to explore new situations in order to improve the intrinsic reward, wasting time that could have been spent gathering more points. This result is not surprising, as the exploration bonus is beneficial when the environment requires significant exploration before provide a reward, which is not the case in this game.

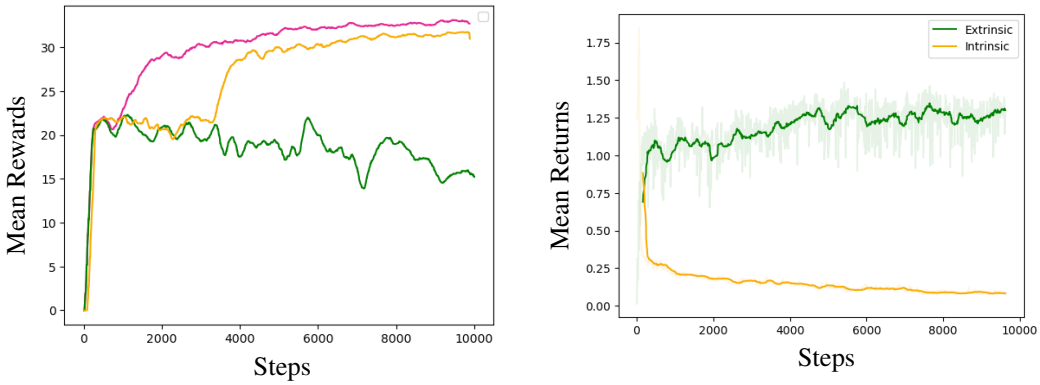

Figure 2: Results of the Freeway RND implementation. On the left, the Rewards (extrinsic only) calculated as the mean of the last 100 ended episodes. On the right, the extrinsic Returns R_E and the intrinsic Returns R_I, both calculated as the mean of the seeds returns on the current step.

[3]More details can be found in the Gymnasium Documentation

3.4.2 Venture

Venture is an Atari game that requires a lot of exploration in order to achieve a good score. There are many rooms to explore, and many enemies. In the PPO paper this game is tested alongside other Atari games, resulting in a mean final score of 0, even if some of the seeds got some points in the initial stages of training. On the contrary, the RND paper reports an high final score, characterized by a rapid improvement in the early stages of training and a steady improvement thereafter. Due to the discussed constraint, only 32 parallel environment are used and the total steps are reduced to 8 thousand, also, we could test only one seed, the 3047.

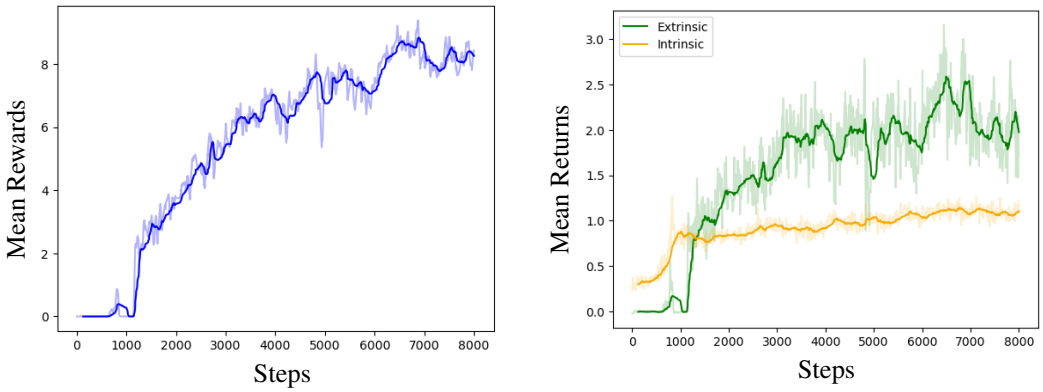

Figure 3: Results of the Venture RND implementation. On the left, the Rewards (extrinsic only) calculated as the mean of the last 100 ended episodes. On the right, the extrinsic Returns R_E and the intrinsic Returns R_I, both calculated as the mean of the current step.

As shown in Figure 3, both the extrinsic and intrinsic returns are increasing. The intrinsic returns indicate that the agent is exploring the environment, while the extrinsic returns show that the agent is also accumulating points.

We tested the trained agent playing Venture with the rendering enabled. Unfortunately, the agent did not perform well in terms of obtaining a high final score. It would explore different rooms and occasionally eliminate enemies, but it was not yet capable of achieving a good score.

We expected this result: the train was conducted on a significantly smaller scale compared to the original work, both in terms of parallel environment and time. However, the main purpose of this experiment was to verify the correctness of the RND implementation, and the obtained results seems to confirm that. Therefore, the experiment can be considered successfully concluded.

4 Link to external resources:

- EnvPool Documentation
- Gymnasium Documentation
- The project's Git-Hub repository.

References

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017a.

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

Mnih V., Kavukcuoglu K., and Silver D. et al. Human-level control through deep reinforcement learning. *Nature*, 2016. doi: 10.1038/nature14236.