Playing Pokemon Red on GameBoy with PyBoy using Proximal Policy Optimization

Domonkos Kertész
Reinforcement Learning
DM887
University of Southern Denmark
doker24@student.sdu.dk

Abstract

The goal of my project is to apply Proximal Policy Optimization, on parallel running, vectorized environments, built upon the library PyBoy.

1 Description

The goal of my project is to apply Proximal Policy Optimization, on parallel running, vectorized environments, built upon the library PyBoy. The project itself can be found on my GitHub `https://github.com/KihiraLove/ReinforcementLearning/tree/main/PokemonRedPPO`.

1.1 Development Steps

1. Setting up additional data files for the game
2. Setting up PyBoy to load the game and additional files, with the correct configuration
3. Wrap the PyBoy in a custom Environment that inherits OpenAIs Gym Env
4. Vectorize the environments with Gymnasium
5. Implement PPO
6. Implement additional functions to process result data

2 The Environment

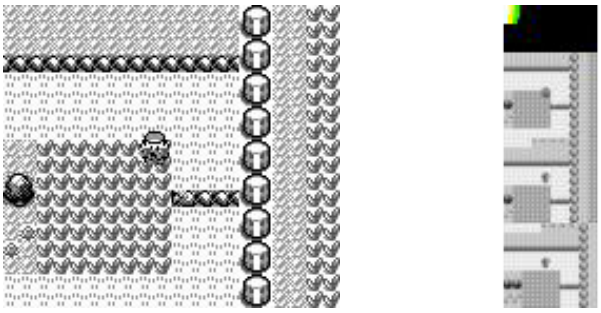
The environment is based on the Python library clled PyBoy, a GameBoy emulator, that lets me interract with the emulated game, using python. The game in question, PokemonRed, comes from an original GameBoy cartridge, which it was copied from, and was saved into a .gb file. This file is passed to the emulator to run.

During run my code can interract with the game using the PyBoy API, which lets me use the 6 buttons, that can be used to interract with the game (**A, B, LEFT, UP, RIGHT, DOWN**) and get the current screen from the emulator. While the game itself is continous, the program only observes the screen every 24 frames. This screen data is the aggregated with the last 2 screen data, and auxiliary data (namely the current hitpoints, total level, and exploration progress) as visual status bars, as it can be seen on Figure 1-3, to form a simple short term memory, this data is the fed to the PPO algorithm.

The way to get information from the game without having to process data from the screen, is to read it from the memory. The memory of PokemonRed is statically allocated, and the game has been reverse engineered, so the memory addresses for the specific data my program needs can be found online.

The game requires a save file, we either need to supply it with one, or it will automatically create one.

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



(a) Emulator screen (b) The algorithms input

Figure 1: The algorithm exploring the map

(a) Emulator screen (b) The algorithms input

Figure 2: The algorithm battling with pokémon

(a) Emulator screen (b) The algorithms input

Figure 3: The algorithm interracting with menus

The emulator is then wrapped by the environment class, that inherits from the Env class of OpenAIs Gymnasium. This way my environment functions the same as a Gym environment would. This way the enviroments can easily be vectorized by Gymnasiums `SyncVectorEnv()` function.

2.1 Reward Structure

The game itself is complex, it includes an interractable environment, interractable non-player characters, navigatable menus, different screens (overworld, battle), that all needs to be used in combination to progree the game, and reach the goal state. Therefore the reward stucture is required to be quite complex as well, the rewards can be seen on Table 1. On the figure, n is the rewards scale, which is 1 by default.

The algorithm can score rewards by exploring the map, this is done by comparing the current screen with the previous screen, and if the difference is great enough, reward is given to the algorithm. Other ways to obtain rewards are, participating in events, these are rare unique occasions in the

Table 1: Reward Structure

Name	Amount	Description
Event	4n	Number of events participated in
Level	4n	Level of gathered pokémons
Heal	4n	Total healing amount
Opponent Level	4n	Largest enemy level
Badge	20n	Number of gathered badges
Explore	1n	Number of new screens discovered

game, catching new pokémon and leveling up pokémon, healing pokémon (replenishing health points after battle), fighting opponents, and receiving badges, which are handed out after beating certain opponents.

2.2 Start and Goal state

The game has a tutorial, which is quite simple for human players.

1. Input player name
2. Interract with NPCs (non-player characters)
3. Chose starting pokémon
4. Travel between two locations to bring an item to an NPC

However, these tasks would require either hardcoding certain steps, or modifying the reward structure. To avoid teaching the algorithm a quite linear sequence of events, I have opted to use a save file, `has_pokedex_nballs.state`, this is a save file which is loaded for every episode as starting point, where the player has already finished the tutorial, has a starter pokémon, and a small number of pokéballs, which are used throughout the game.

Similarly, the game is technically endless, the story ends at some point, but the game is playable after. Reaching the ending is a complex goal, which would require thousands of hours of training, therefore I opted to chose an arbitrary ending, which is to obtain the first badge, altough the algorithm never managed to reach this goal.

3 Proximal Policy Optimization

Since OpenAI released it's paper on PPO in 2017, it has became one of the most popular reinforcement learning algorithm, and for good reason, as PPO is a reliable and efficient method for training policies on complex environments, such as mine.

3.1 Concept

PPO is a policy based method, where the objective is $\pi(a \mid s; \theta)$, where `s` are states, that are mapped to `a` actions, parameterized by θ. It optimizes a surrogate objective function that approximates the true objective of maximizing expected cumulative rewards. This objective function includes a ratio between the new policy and the old policy. One of the main innovations in PPO is the introduction of a clipping mechanism in the objective function. The clipped objective ensures that the new policy does not deviate too much from the old policy, thereby preventing large updates that can destabilize training. The clipped objective is given by:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t,\ \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \tag{1}$$

Where $\pi_\theta(a_t \mid s_t)$ is the new policy, $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ is the old policy.

A_t is the advantage function, and ϵ is the hyperparameter that controls the clipping range. The advantage function

$$A_t = Q(s_t, a_t) - V(s_t) \tag{2}$$

where:

- $Q(s_t, a_t)$ is the action-value function, representing the expected return for taking action a_t in state s_t.
- $V(s_t)$ is the value function, representing the expected return for being in state s_t.

3.2 Steps of PPO

- Interact with the environment to collect a set of trajectories (state-action-reward sequences) using the current policy.
- Compute the returns (cumulative rewards) for each trajectory and compute the advantages using the value function to reduce variance.
- Perform multiple epochs of optimization on the collected data and use the clipped surrogate objective to update the policy parameters, ensuring the updates are not too large.
- Simultaneously, update the value function to improve the baseline estimation.

3.3 Pseudocode

Algorithm 1 Proximal Policy Optimization (PPO)

```
Initialize policy network π_θ and value network V_θ with random weights
for iteration = 1, 2, ..., N do
    Collect trajectories by running policy π_θ in the environment
    Compute rewards-to-go (returns) and advantages for each timestep
    for epoch = 1, 2, ..., K do
        for each mini-batch of data do
            Compute the surrogate loss L^CLIP(θ)
            Compute the value function loss
            Optimize the policy network using the surrogate loss
            Optimize the value network using the value function loss
        end for
    end for
end for
```

3.4 Advantages of PPO

The pseudocode for Proximal Policy Optimization (PPO) outlines a structured approach to training reinforcement learning agents.

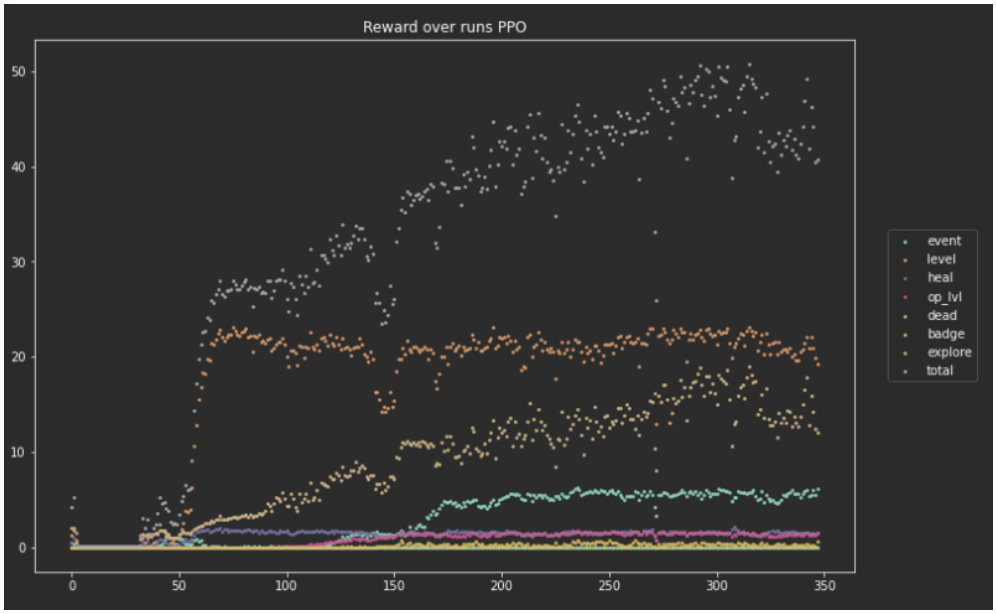
Initially, the policy network π_θ and value network V_θ are initialized with random weights. The algorithm operates over multiple iterations, where in each iteration, trajectories (sequences of states, actions, and rewards) are collected by executing the current policy in the environment.

Following trajectory collection, the rewards-to-go (returns) and advantages for each timestep are computed to evaluate the quality of actions taken.

The policy and value networks are then optimized over several epochs. During each epoch, the data is divided into mini-batches, and for each mini-batch, the surrogate loss $L^{\text{CLIP}}(\theta)$ is calculated to update the policy network.

This loss function incorporates a clipping mechanism to prevent large updates that could destabilize training.

The value function loss is computed and used to optimize the value network, enhancing its ability to estimate future rewards. This iterative process continues, progressively improving the policy and value networks to achieve better performance in the environment.



(a) Rewards over 350 episodes

```
step:   2421 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 12.60 sum: 12.60
step:    426 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  0.96 sum:  0.96
step:    999 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  3.24 sum:  3.24
step:   1420 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  6.48 sum:  6.48
step:   2108 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 12.06 sum: 12.06
step:   1810 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  9.66 sum:  9.66
step:   3511 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 13.14 sum: 13.14
step:    365 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  1.08 sum:  1.08
step:   2474 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 10.68 sum: 10.68
step:    703 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  3.12 sum:  3.12
step:   2617 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 13.02 sum: 13.02
step:    574 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  3.66 sum:  3.66
step:    884 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  5.16 sum:  5.16
step:   3453 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 12.48 sum: 12.48
step:   2217 event:  0.00 level:  4.00 heal:  1.92 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  8.88 sum: 14.80
step:    382 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  3.72 sum:  3.72
```

(a) Early in training

```
step:   2053 event:  4.00 level:  4.00 heal:  2.04 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 14.76 sum: 24.40
step:    843 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 11.40 sum: 11.00
step:    672 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  8.52 sum:  8.52
step:   2903 event:  0.00 level: 16.00 heal:  1.87 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 19.74 sum: 37.21
step:    816 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  7.62 sum:  7.62
step:   3363 event:  4.00 level:  8.00 heal:  0.40 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 19.50 sum: 31.50
step:   1188 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore: 11.28 sum: 11.28
step:   1703 event:  4.00 level:  4.00 heal:  0.67 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 16.38 sum: 24.65
step:   1348 event:  0.00 level:  0.00 heal:  0.00 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 11.28 sum: 10.88
step:   2739 event:  4.00 level: 24.00 heal:  1.03 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 19.14 sum: 47.77
step:   3142 event:  0.00 level:  4.00 heal:  3.27 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 20.64 sum: 27.51
step:   2752 event:  4.00 level: 12.00 heal:  2.37 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 18.90 sum: 36.87
step:   2953 event:  0.00 level: 16.00 heal:  2.86 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 18.54 sum: 37.00
step:   3580 event:  0.00 level: 20.00 heal:  1.21 op_lvl:  0.00 dead: -0.80 badge:  0.00 explore: 21.00 sum: 41.41
step:   5067 event:  0.00 level: 16.00 heal:  0.01 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 23.88 sum: 39.49
step:   3105 event:  4.00 level: 24.00 heal:  1.03 op_lvl:  0.00 dead: -0.40 badge:  0.00 explore: 19.44 sum: 48.07
step:   2954 event:  0.00 level: 12.00 heal:  1.92 op_lvl:  0.00 dead: -0.80 badge:  0.00 explore: 18.96 sum: 32.08
step:    942 event:  0.00 level: 12.00 heal:  2.81 op_lvl:  0.00 dead: -0.00 badge:  0.00 explore:  9.60 sum: 24.41
```

(b) Later in training

Figure 4: Collected rewards during training

4 Results

While the implemented algorithm and environment are quite unstable, and training requires a lot more time and computational resources, than I originally tought it would, I was able to gather some results by running smaller experiments.

The scale my misjudgement of the training time and resources is severe, solving this task would require thousands of hours of training even on a server with many processor cores, therefore my results training with limited time and on a personal computer, are quite humble. But the algorithm showed progrees, even if small, completed tasks and obtained rewards, better in each episode.

5 Conclusion

This task would require much better resources and more time, but I am personally happy with what I have achieved.

time/	
fps	309
iterations	3
time_elapsed	397
total_timesteps	122880
train/	
approx_kl	0.00792137
clip_fraction	0.0861
clip_range	0.2
entropy_loss	-1.78
explained_variance	0.754
learning_rate	0.0003
loss	-0.0133
n_updates	6
policy_gradient_loss	-0.00821
value_loss	0.00212

(a) Early in training

time/	
fps	276
iterations	22
time_elapsed	3261
total_timesteps	901120
train/	
approx_kl	0.016192868
clip_fraction	0.189
clip_range	0.2
entropy_loss	-1.5
explained_variance	0.757
learning_rate	0.0003
loss	6.98e-05
n_updates	63
policy_gradient_loss	-0.00818
value_loss	0.00163

(b) Later in training

Figure 5: Stats of the training