Evaluation of Proximal Policy Optimization Algorithms

Shansong (Sam) Huang, Qinyun (Peter) Yu

April 27th, 2022

DS8010: Interactive Learning in Decision Process

Introduction

Reinforcement learning occurs under a trial-and-error process that helps model reward-based learning. In the 20th century, there have been a number of different approaches that aim to solve the Bellman equation that fall under value function approximation algorithms, with the drawback that these methods are considered tabular [Sutton and Barto, (2020)].

In the 21st century, reinforcement learning was revitalized when Krizhevsky et al., published a paper on the "*ImageNet Classification with Deep Convolutional Neural Networks*". This paper showcased the ability to couple graphical processing units with neural networks and backpropagation, and resulted in the 2015 paper by DeepMind, titled as "*Human-level Control Through Deep Reinforcement Learning*". This paper allowed neural networks to perform Q-Learning, a traditional approach, to learn to play Atari games through Deep Q-Learning Networks (DQN) [Mnih et al, (2015)] . That being said, there are a couple issues with DQNs (e.g., convergence issues and instability), which led to policy gradient methods being popularized.

For policy gradient methods, according to Peters (2010), by parameterizing a policy, one could instead directly learn about a policy rather than approximating through V- or Q-functions and the policy could be updated through gradient ascent. In 2015, a paper titled "Trust Region Policy Optimization" (TRPO) was published where the TRPO method addressed DQN instability, while also not sacrificing out too much on the data efficiency. Essentially, TRPO worked very well having addressed some of the many shortcomings of policy gradient methods and allowed researchers to apply the algorithm to real life scenarios [Schulman et al., (2015)]. Unfortunately, however, the algorithm itself was simply very complicated to understand and very difficult to implement.

Due to the difficulties implementing TRPO, Proximal Policy Optimization (PPO) was published. This new algorithm was inherently very similar to the previously published TRPO paper, however given slight modifications, PPO was simpler, faster, and much more data efficient [Schulman et al., (2017)]. Given the benchmarks from Schulman et al., it was state of the art at the time, however from the our perspective and experience, it is clear that PPO, given the niche of an on-policy method, only is considered the optimal agent, when handling environments that lack parallelization, and do not have a environment model. When the aforementioned exist for the environment, there exists better models like MuZero by Schrittwieser et al., and SAC by Haarnoja et al.

Literature Review

Invention of PPO

In 2015, human-level control was attained over a number of Atari game environments through Deep Q-Learning Networks (DQN). That being said, as DQNs contained function approximators, bootstrapped, and learn off-policy, it suffers from divergence issues. [Mnih et al, (2015)]. What this means is for certain problem spaces, DQNs were unstable, and alternatives were needed.

A known alternative were Policy Gradient algorithms which directly parameterized the policy and used randomly instantiated weights to try to maximize value under such parameterized policy, given a state, whereby gradient ascent updates were then performed [Peters, (2010)]. While simpler said, vanilla policy gradient approaches still had a number of drawbacks. Some of its main drawbacks include finding second-order derivatives which has horrible scalability, as well as high computational complexity. These vanilla policy gradient methods also suffered from poor data efficiency and lacked robustness. One idea that did transfer over to TRPO and PPO was the idea of reducing variance in gradient estimates, through an advantage function. This estimates how good an action is compared to average action as a baseline when given a specific state. The idea was to reduce the high variance in the gradient estimates between the old policy and the new policy, which thereby would increase the stability of the vanilla policy gradient algorithm.

In 2015, the Trust Region Policy Optimization (TRPO) algorithm was published, which addressed many of the shortcomings of vanilla policy gradient methods. The TRPO algorithm added KL divergence constraints for enabling the trust-region for the optimization process. In the trust region, the search for the local minimum or local maximum is bounded in that region, and the local minimum or local maximum determines the direction of the optimal point. Where as in TRPO, it added KL divergence constraints so that the newly updated policy would not be too far away from the old policy, hereby lowering the policy update deviation [Karunakaran, (2020)]. However, the TRPO algorithm was relatively complicated to implement and understand, and was not compatible with architectures that included noise or parameter sharing. For many, implementing the TRPO algorithm could prove very challenging. Therefore, the Proximal Policy Optimization algorithm was then introduced that maintained the data efficiency and reliable performance of TRPO, while using only first-order optimization [Karunakaran, (2020)].

Notations

- $\hat{E}_t$ denotes the empirical expectation over a finite batch of samples
- π_θ is a stochastic policy parameterized by θ
- $\hat{A}_t$ = Q(s,a) - V(s), $\hat{A}_t$ is an estimator of the advantage function at time step t
- V(s) is the State-value function measures the expected return of state s
- $Q(s,a)$ is the Action-value function. It assesses the expected return of a pair of state and action (s,a)

Policy Optimization

Policy Gradient Methods

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The most commonly used gradient estimator with advantage function has the form:

$$\hat{g} = \hat{E}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

Trust Region Policy Optimization (TRPO)

In order to avoid parameter updates that change the policy too much at one step to improve training stability, the TRPO introduce a KL divergence constraint on the size of policy update at each iteration.

$$\underset{\theta}{maximize} \quad \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right] \tag{2}$$

$$subject\ to \quad \hat{E}_t\left[KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \leq \delta \tag{3}$$

KL(Kullback-Leibler) divergence measures the distance between old and new probability distribution. θ_{old} is the vector of policy parameters before the update.

$$D_{KL}(\pi_{old}||\pi) = \int_x \pi_{old}(x) \log \frac{\pi_{old}(x)}{\pi(x)} dx \tag{4}$$

TRPO maximizes the objective function and subjects to the trust region constraint which enforces the distance between old and new policies measured by KL-divergence to be small enough, within a parameter of δ. However, this makes TRPO a complicated algorithm to implement. The KL constraint adds additional overhead in the form of hard constraints to the optimization process.

Proximal Policy optimization (PPO)

Given that TRPO is relatively complicated, here comes the PPO which has a much simpler approach. PPO simplifies TRPO by using a clipped objective function while retaining similar performance. First, similar to TRPO objective function, it defines the probability ratio between the new policy and old policy as $r(\theta)$ [Schulman et al, (2017)].

$$r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{5}$$

Then, the objective function of TRPO becomes:

$$L^{CPI}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right] = \hat{E}_t[r_t(\theta)\hat{A}_t] \tag{6}$$

CPI refers to conservative policy iteration. Without a constraint, maximization of L^{CPI} would lead to an excessively large policy update and big policy ratios. Next, PPO imposes the constraint by forcing $r(\theta)$ to be within a small interval around 1, [1 - ϵ, 1 + ϵ], where ϵ is a hyperparameter.

The main objective PPO proposed:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{7}$$

The first term inside the min function is L^{CPI}. The second term clip$(r(\theta), 1-\epsilon, 1+\epsilon)$ clips the ratio $r(\theta)$ to be within $[1-\epsilon, 1+\epsilon]$. By taking the minimum of the clipped and unclipped objective we can have the final objective represent a lower bound on the unclipped objective [Schulman et al, (2017)]. A positive $\hat{A}_t$ indicates the action taken is good, otherwise if $\hat{A}_t$ is negative, the action taken by the agent is bad[Karunakaran, (2020)]. For both cases, with $L^{CLIP}(\theta)$ objective function, the PPO ensures there is no large policy change.

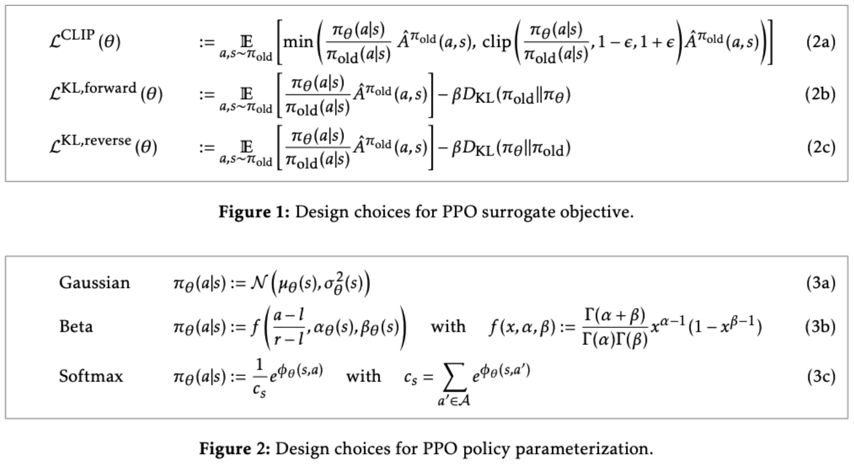
Revisiting PPO

Proximal Policy Optimization Algorithms was published in 2017. Since then, PPO has been a popular deep policy gradient algorithm. However in the later paper ”Revisiting Design Choices in Proximal Policy Optimization” [Hsu et al., 2020], two design choice in PPO are revisited, they are (1) clipped probability ratio for policy regularization and (2) parameterize policy action space by continuous Gaussian or discrete softmax distribution.

Standard PPO means that the common combination of clipped surrogate objective with continuous Gaussian and discrete Softmax policy parameterization. Three failure modes in PPO were found:

- 1: On continuous action spaces, standard PPO is unstable when rewards vanish outside bounded support. This failure mode illustrates a scenario where the clipped objective fails to recover from a bad Gaussian policy update step. The clipping mechanism effectively prevents the policy from moving further away once it is outside the trust region, but it does not bound the size of an individual policy update step. This is problematic in the case that a single reward signal can cause the policy to end up in regions with low reward signal.
- 2: On discrete action spaces with sparse high rewards, standard PPO often gets stuck at suboptimal actions. This failure mode illustrate a scenario where clipping converges to suboptimal actions for distrecte action spaces with the standard softmax policy parameterization. This failure mode is problematic because it happens within the convergence regime of the more principled KL-regularized PPO which clipping aims to approximate.
- 3: The policy is sensitive to initialization when there are locally optimal actions close to initialization. This is the case that standard PPO can converge to suboptimal actions, and even close to the initialization.

From ”Revisiting Design Choices in Proximal Policy Optimization” [Hsu et al., 2020], the paper also proposed improvements for three PPO failure modes. The paper propose that discretizing the action space or use Beta distribution 3b helps avoid failure mode 13 associated with Gaussian policy. Also, using KL regularization 2b and 2c as an alternative surrogate objectives to the clipped surrogate objective 2a helps resolve failure mode 1 and 2.



$$\mathcal{L}^{\text{CLIP}}(\theta) := \mathop{\mathbb{E}}_{a,s\sim\pi_{\text{old}}}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\hat{A}^{\pi_{\text{old}}}(a,s),\ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_{\text{old}}}(a,s)\right)\right] \quad (2a)$$

$$\mathcal{L}^{\text{KL,forward}}(\theta) := \mathop{\mathbb{E}}_{a,s\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\hat{A}^{\pi_{\text{old}}}(a,s)\right] - \beta D_{\text{KL}}(\pi_{\text{old}}\|\pi_\theta) \quad (2b)$$

$$\mathcal{L}^{\text{KL,reverse}}(\theta) := \mathop{\mathbb{E}}_{a,s\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\hat{A}^{\pi_{\text{old}}}(a,s)\right] - \beta D_{\text{KL}}(\pi_\theta\|\pi_{\text{old}}) \quad (2c)$$

Figure 1: Design choices for PPO surrogate objective.

Gaussian $\pi_\theta(a|s) := \mathcal{N}\left(\mu_\theta(s), \sigma_\theta^2(s)\right)$ (3a)

Beta $\pi_\theta(a|s) := f\left(\frac{a-l}{r-l}, \alpha_\theta(s), \beta_\theta(s)\right)$ with $f(x,\alpha,\beta) := \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x^{\beta-1})$ (3b)

Softmax $\pi_\theta(a|s) := \frac{1}{c_s}e^{\phi_\theta(s,a)}$ with $c_s = \sum_{a'\in\mathcal{A}} e^{\phi_\theta(s,a')}$ (3c)

Figure 2: Design choices for PPO policy parameterization.

Figure 1: [Hsu et al., 2020]

Phasic Policy Gradient

Phasic Policy Gradient(PPG) is a reinforcement learning framework which modifies traditional on-policy actor-critic methods such as PPO by separating policy and value function training into distinct phases [Cobbe et al., 2020]. Compared to PPO, the PPG was found significantly improves sample efficiency on the challenging Procgen Benchmark [Cobbe et al., 2020].

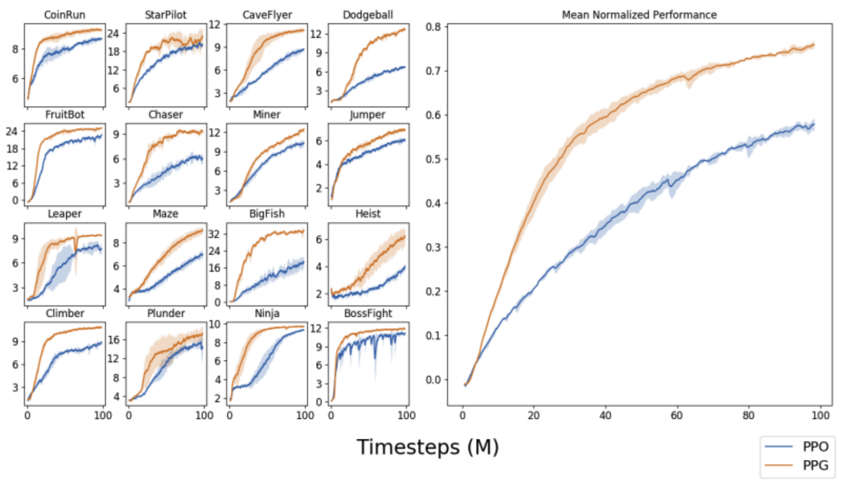


Figure 2: Sample efficiency of PPG compared to a PPO baseline [Cobbe et al., 2020]

The PPO algorithm rely on the actor-critic framework, and the policy and the value function. There are advantages and disadvantages of sharing parameters between the policy and the value function. One advantage is that features trained by each objective can be used to better optimize the other. However, the disadvantages is that it is not clear how to appropriately balance the competing objectives of the policy and the value function.Any method that jointly optimizes these two objectives with the same network must assign a relative weight to each. Regardless of how well this hyperparameter is chosen, there is a risk that the optimization of one objective will interfere with the optimization of the other. Second disadvantage is that it demands the same data for training two networks at the same time.

To address those problems, PPG modifies the on-policy actor-critic policy gradient algorithm PPO to have seperate training phases for policy and value functions. Training proceeds in two alternating phases:

- Policy phase: Train the agent with PPO. Specifically, train the policy network using the clipped surrogate objective L^{clip}.
- Auxiliary phase: optimize the policy network with a joint objective that includes an arbitrary auxiliary loss and a behavioral cloning loss:

$$L^{joint} = L^{aux} + \beta_{clone} \cdot \hat{E}_t \left[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \tag{8}$$

$$L^{aux} = \frac{1}{2} \cdot \hat{E}_t \left[(V_{\theta_\pi}(s_t) - \hat{V}_t^{targ})^2\right] \tag{9}$$

- V_{θ_π} is an auxiliary value head of the policy network
- β_{clone} is a hyperparameter for controlling the trade-off of how much we would like to keep the policy not diverge too much from its original behavior while optimizing the auxiliary objectives.
- N_π controls the number of policy updates performed in each policy phase
- E_π and E_V control the sample reuse for the policy and value function respectively, during the policy phase item E_{aux} controls the sample reuse during the auxiliary phase, representing the number of epochs performed across all data in the replay buffer

Improvements on PPO-Clip Algorithm

In the process of one-step updating, PPO-Clip tend to have less computations than PPO-KL, therefore the convergence time of PPO-Clip tend to shorter than PPO-KL, and is more stable than PPO-KL. Therefore the PPO-Clip has received more attention, many research has proposed improvements on the PPO-Clip algorithm. Here we mainly summarize two improvements paper based on the PPO-Clip algorithm.

1. Truly Proximal Policy Optimization

The paper "Truly Proximal Policy Optimization"[Wang et al., 2020] showed that PPO's optimization behavior is still far from being fully understood.

Firstly, the PPO could not restrict the likelihood ratio as it attempts to do. In general, PPO could generate an effect of preventing the likelihood ratio from exceeding the clipping range too much, but it could not strictly bound the likelihood ratio.

Secondly, the PPO could not enforce a well-defined trust region constraint. Under the state-action (s_t,a_t), if the likelihood ratio $r_t(\theta)$ is not bounded, then neither could the corresponding KL divergence be bounded. Therefore, together with the first point, PPO could not bound KL divergence.

From the two issue above, the PPO may suffer from the risk of performance instability. To address this issue, the paper present an enhanced PPO method, the Truly PPO. Truly PPO is formed by combining Trust Region-based PPO (TR-PPO) and PPO with Rollback (PPO-RP), where PPO-RP is intend for solving first issue and TR-PPO is intend for the second issue. The results showed that Truly PPO improves PPO by: 1)adopting a new clipping function to support a rollback behavior to restrict the difference between the new policy and the old one; 2) the triggering condition for clipping is replaced with a trust region-based one, such that optimizing the resulted surrogate objective function provides guaranteed monotonic improvement of the ultimate policy performance. The new algorithm Truly PPO improves the original PPO on both sample efficiency and performance. The TR-PPO and Truly PPO achieved higher reward than PPO on Atari tasks.

2. Proximal Policy Optimization with Relative Pearson Divergence

Proximal Policy Optimization with Relative Pearson Divergence [Kobayashi, 2021] proposed a PPO algorithm based on relative Pearson divergence, and it performed as well as or better than clip-PPO. PPO clips density ratio of the latest and baseline polices with a threshold, while its minimization target is unclear. [Kobayashi, 2021] proposed a new variant of PPO

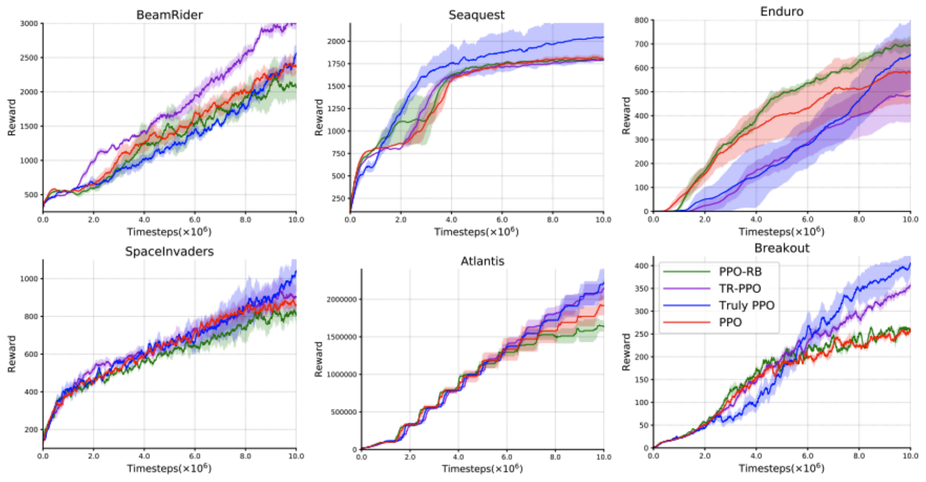


Figure 3: Episode rewards of the policy during the training process on Atari Tasks. [Wang et al., 2020]

by considering a regularization problem of relative Pearson(RPE) divergence, therefore it is called PPO-RPE. PPO-RPE performed as well as or better than the conventional methods in terms of the task performance by the learned policy, where the conventional methods includes PPO and PPO-RB.

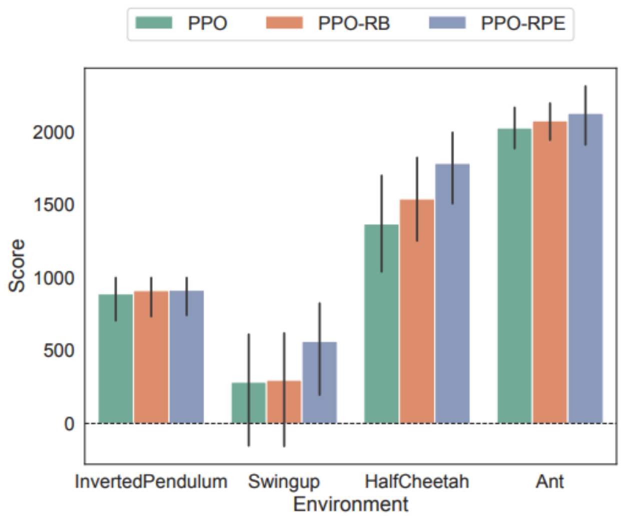


Figure 4: Summary of four benchmark tasks as bar plots [Kobayashi, 2021]

Methods

Using the Python OpenAI Gym environment, we plan on implementing the Proximal Policy Optimization algorithm as follows below in PyTorch and benchmarking our implementation of PPO on several Atari Game Environments. Due to the lack of computational resources, a limited number of training episodes were given to various Atari games, especially since the original PPO paper by Schulman et al., trained their PPO algorithm on 10 million episodes (40 million frames). Here are some of the other algorithm implementation ideas taken from Schulman et al.,

Actor-Critic Method

For training purposes, two deep neural network models are used - the actor, and the critic model. The actor and critic model contain similar architectures and differ in which the actor, when given a state, returns the possible actions of the state, while the critic estimates and evaluates the actions of the actor given a policy.

Essentially, the actor will take the pre-processed frames of the Atari environment, and using a softmax activation function, will return a list of action probabilities. By sending a list of actions to the environment, we track the rewards obtained. If a reward is obtained due to a predicted action, that feedback is sent to our critic model.

A slight modification is made where only one actor is used rather than the many actors in the Schulman et al., paper.

Algorithm 1 PPO Actor-Critic Algorithm

```
  1: Input: initial policy parameter $\theta_0$, initial value function parameters $\phi_0$
  for <k = 0,1,2,...  > do
      2: Collect set of trajectories $D_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
      3: Compute rewards-to-go $\hat{R}_t$
      4: Compute advantage estimates, $\hat{A}_t$ based on the current value function $V_{\phi k}$
      5: Update the policy by maximizing the PPO-Clip objective:
      Maximize $L^{CLIP+VF+EB} = L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 L^{EB}(\theta)$
  end for
```

Step-by-step:

- 1: Initialize the Actor and Critic networks and parameter θ_0 and ϕ_0
- 2: Collect a batch of trajectories from the newest Actor policy
- 3: Compute the reward for each trajectory in each step
- 4: Compute the estimated advantage for each trajectory from the newest Critic network
- 5: Optimize the loss function with respect to the actor-critic parameters. This is done by calculating the combined loss of the policy, value, and entropy bonus (in respective order below).

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{10}$$

$$V^{\pi_0}_{CLIP} = clip(V^{\pi_0}(s_t) - \hat{V}_t, -\epsilon, +\epsilon) \tag{11}$$

$$L^{VF}(\theta) = \frac{1}{2}E\left[max\left((V^{\pi_\theta}(s_t) - R_t)^2, (V^{\pi_\theta}_{CLIP}(s_t) - R_t)^2\right)\right] \tag{12}$$

$$L^{EB}(\theta) = E\left[S[\pi_\theta](s_t)\right] \tag{13}$$

- 6: Updating the Critic's parameters by gradient descent on using the custom loss function as described in Schulman et al.,

Pre-processing and Frame Stacking

In past literature (namely with DeepMind's papers), pre-processing and frame stacking is performed on the Atari environment before being handled by the respective algorithms [Mnih et al., 2013,2015]. In terms of pre-processing, each of the game frames are downsampled from a height and width of 210x160 to 84x84. This is to ensure that our inputs for our actor and critic model, which utilize convolutional layers, are not too complicated as well as still containing similar levels of information. According to [Mnih et al., 2013,2015] as well as in our implementation, 4 frames are taken and fed into our actor-critic model as well, essentially where our PPO agent will base their next action based on the past 4 sequence of actions. This makes sense as for example, in Space Invaders, if aliens are moving slow, then the player has time to perform other actions, while if the aliens are fast, then the player knows to prioritize certain actions.

Experimental Setup

Due to the lack of number of computational resources when using Google Colaboratory, a very limited number of episodes are used to train our PPO agent. This number varied based on the type and prior knowledge of the expected length of game. For that purpose, SpaceInvaders-v0, Breakout-v0, Centipede-v0 were trained on 150 episodes, while VideoPinball-v0 and KungFuMaster-v0 were trained on 10.

In addition, in the past couple years, the Gym environment removed Atari games and there was a bit of a trouble setting up the Atari environment. To simplify the research a bit, no hyperparameter tuning was conducted and most of the hyperparameters were taken straight from Schulman et al., in an effort to try to match initial results as shown in table 1. The only places of difference are with the number of actors, where the environment can be parallelized and an actor can be assigned to its own environment. To avoid potential confusion (especially since it is hard to distinguish erroneous implementations), only one actor was used. The clipping parameter when compared to Schulman et al., and the Adam stepsize were slightly modified.

Hyperparameter	Value
Horizon	128
Adam stepsize	1E-4
Num. epochs	3
Minibatch size	32
Discount (γ)	0.99
GAE parameter (λ)	0.95
Number of actors	1
Clipping parameter (ϵ)	0.1
VF coeff.	1
Entropy coeff.	1E-3

Table 1: PPO hyperparameters used in Atari experiments

Results

It is clear that in such a limited time frame, not many conclusive results can be obtained from our PPO implementation (Figure 5). While it seems that in KungFuMaster-v0, and Centipede-v0 there are slight improvements when compared with the initial episodes, it seems that such improvements are minimal.

When comparing the results of the original Schulman et al., paper with our results, it is clear that Schulman et al., Atari scores are vastly superior to the results we achieved. This can be attributed to the lack of hyperparameter tuning done as the original researchers used Mujoco, a physics engine, to tune the model hyperparameters (clipping, GAE, coefficients), as well as researchers having access to more powerful machines. Given the limited time frame, much lower score results are obtained.

Game	A2C	ACER	PPO	Our PPO Estimation
Breakout	303.0	456.4	274.8	1.35
Centipede	3496.5	8904.8	4386.4	2047.1
VideoPinball	19735.9	156225.6	37389.0	29904.9
SpaceInvaders	744.5	1213.9	942.5	212.5
KungFuMaster	24900.3	27599.3	23310.3	752.3

The table shows the mean performance. The value of A2C, ACER, PPO is from comparison of PPO and A2C on all 49 ATARI games of paper ”Proximal Policy Optimization Algorithms” [Schulman et al., 2017].

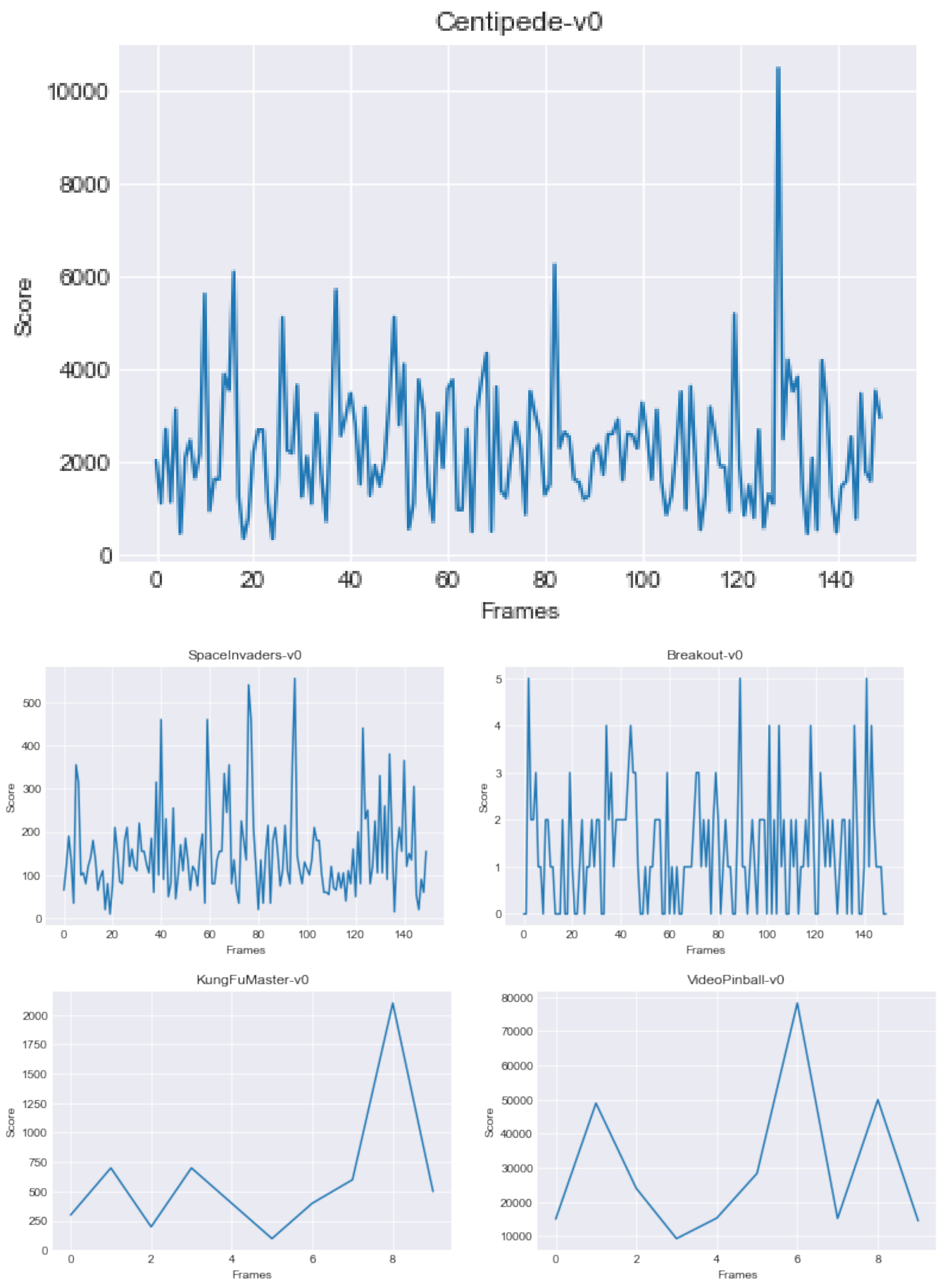


Figure 5: Results of our PPO implementation on various Atari Environments

Conclusions

Main Finding

It is clear that in the limited time space that no conclusive benchmarks can be made comparing our implementation of PPO to Schulman et al.'s implementation, nor their provided literature results of the A2C or ACER benchmark Atari scores. This is especially evident when considering that the paper trained over 10 million episodes over 8 actors on the Atari gameset, which is significantly slower than previously done course-base work as the Atari environments needs to be analyzed and classified using Convolutional Neural Networks.

In terms of the findings when compared with the original paper, it is clear that tricks are needed to speed up policy finding, as even simple games like Breakout (the game where you bounce a ball against a wall to escape) require millions of episodes of training. One of such tricks that the authors used was multiple actors. Given the highly parallelizable nature of Atari games (which does not reflect the real world) rather than 1 actor like we used, we could use multiple actors to try to speed up exploration and come to an optimal policy sooner.

Another trick that could have been used is frame composition. In Mnih et al.'s 2015 paper, rather than using every four frames to input into their Actor-Critic network, a frame-skipping parameter exists. This is because given these frames are overlapping, we capture the limited motion of the Atari games much easier, as frames that are irrelevant to capturing motion are hereby removed. If we implemented this trick, this would have sped up our network training and resulted in a sooner optimal policy.

Another simple finding that turned out to be crucial was that given a near-optimal policy, the player would then have longer gameplay before terminal state (death). This means that our expectations for training time should have been much extended, as initially when random weights are instantiated for the models, training took a mere seconds, however as the days went on, training took significantly longer. Due to this reason, it was looking evident that parallelization (which is not available in every environment, but is for Atari) and improved hardware (in the form of GPUs) were needed. Google Colaboratory was not a good substitute as the intention of Google Colaboratory is to teach users rather than for any significant training, as so, user training time is significantly limited when the Google servers are busy.

When looking back, proximal policy optimization algorithm is miles ahead of brute force solutions, however when considering the purposes of reinforcement learning and what it aims to solve (i.e., reward-based learning), it is evident that when environments begin to grow complex, current reinforcement learning algorithms are not enough and powerful parallelization hardware or better algorithms are still needed.

Study Weaknesses

To highlight some of the weaknesses in our implementation of proximal policy optimization, we found that training time was extensively long. In the Schulman et al., paper, 8 actors were used for training on the Atari Learning Environment. This is because the Atari games can have games running in parallel. This however is not the case in all real life scenarios. For example, if one were to try running PPO on stock market data, there's only one stock market present. For our implementation, more detail should have been paid to multiple actors which would have resulted in more exploration, and in other words, faster optimality.

Another weakness of our study was that the hyperparameters were taken from the original paper, which trained its hyperparameters on Mujoco, a simple physics engine. For the sake of maximizing performance on Atari games, perhaps tuning could have been performed on Atari games, rather than relying on the papers, especially given the resource-intensive nature of training.

One of the inherent weaknesses of reinforcement learning is that there is no tell-tale sign that our implementation is fully correct. With the limited training time, it is especially hard to tell, however with a lot more training time, the mean score and score plot should tell if the implementation is correct.

Finally, to revisit our first study weakness, it needs to be highlighted that proximal policy optimization is only one of the state of the art algorithms in 2017. This was evident in our table results, as the ACER algorithm outperformed PPO in coincidentally all the games we chose. Additionally, that since 2017, a number of different algorithms have taken its throne of state of the art, and that now, PPO serves a niche field of on-policy algorithms that do well for non-parallelized environments. If off-policy data is available, or if the given environment is highly parallelizable, other alternative algorithms exist that outperform PPO. Such algorithms include Soft Actor-Critic, Twin delayed DDPG, MuZero, and Muesli.

Future Research Directions

In terms of the proximal policy optimization algorithm, the main issue is the implementation of multiple actors for our Actor-Critic framework. With such implementation, training would have been sped up significantly. Additionally, improvements on the frame processing can be conducted, as shown by our example choice to not implement DeepMind's frame skipping option, which would have allowed motion to be captured better in our network, resulting in faster training. Finally, more Atari environments need to be evaluated and trained for a much longer duration as only a select few environments were chosen.

In terms of the overall scope of the project, more focus should have been placed on newer algorithms. Atari environments have been the focus for many reinforcement learning papers, and given the recent boom in algorithm development, much more focus should have been placed on algorithms like MuZero by Schrittwieser et al., which are more relevant given their recent crowning as state of the art.

References

Chrysovergis, I. (2021, June 24). Keras Documentation: Proximal policy optimization. Keras. Retrieved April 1, 2022, from https://keras.io/examples/rl/ppo$_{c}artpole/$

Cobbe, K., Hilton, J., Klimov, O., amp; Schulman, J. (2020, September 9). Phasic policy gradient. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/2009.04416

Haarnoja, T., Zhou, A., Abbeel, P., amp; Levine, S. (2018, August 8). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/1801.01290

Hsu, C. C.-Y., Mendler-Dünner, C., amp; Hardt, M. (2020, September 23). Revisiting design choices in proximal policy optimization. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/2009.10897

Karunakaran, D. (2020, December 4). Proximal policy optimization(ppo)- a policy-based reinforcement learning algorithm. Medium. Retrieved April 1, 2022, from https://medium.com/intro-to-artificial-intelligence/proximal-policy-optimization-ppo-a-policy-based-reinforcement-learning-algorithm-3cf126a7562d

Kirzhevsky, A., Sutskever, I., amp; Hinton, G. E. (n.d.). ImageNet classification with deep convolutional ... - neurips. Retrieved April 2, 2022, from https://proceedings.neurips.cc/paper/2012/file/c3
Paper.pdf

Kobayashi, T. (2021, March 15). Proximal policy optimization with relative Pearson divergence. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/2010.03290

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., amp; Riedmiller, M. (2013, December 19). Playing Atari with deep reinforcement learning. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., amp; Hassabis, D. (2015, February 25). Human-level control through deep reinforcement learning. Nature News. Retrieved April 27, 2022, from https://www.nature.com/articles/nature14236

Peters, J. (n.d.). Policy gradient methods. Scholarpedia. Retrieved April 1, 2022, from http://www.scholarpedia.org/article/Policy$_gradient_methods$

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., amp; Abbeel, P. (2017, April 20). Trust region policy optimization. arXiv.org. Retrieved April 1, 2022, from https://arxiv.org/abs/1502.05477

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., amp; Silver, D. (2020, February 21). Mastering Atari, go, chess and shogi by planning with a learned model. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/1911.08265

Sutton, R. S., Barto, A. G. (2020). Reinforcement learning: An introduction. pg 73-78.The MIT Press.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017, August 28). Proximal policy optimization algorithms. arXiv.org. https://arxiv.org/abs/1707.06347

Wang, Y., He, H., Wen, C., amp; Tan, X. (2020, January 14). Truly proximal policy optimization. arXiv.org. Retrieved April 27, 2022, from https://arxiv.org/abs/1903.07940