Proximal Policy Optimization Algorithms: a review

Chagniot Philomène philomene.chagniot@ensae.fr
Delanoue Pierre pierre.delanoue@ensae.fr
Meunier Dimitri dimitri.meunier@ensae.fr

Abstract

We review the article Proximal Policy Optimization Algorithms, written by Shulman et al[13]. PPO belongs to the family of Policy Gradient algorithms, like Advantage Actor Critic [10], and to the subfamily of Trust Region methods, like TRPO [14]. To fully understand how PPO works and how it differs from standard Policy Gradient methods we extensively detail the Advantage Actor Critic, TRPO and PPO algorithms. We also implemented A2C and PPO from scratch and ran experiments to compare the practical performances of these algorithms on the CartPole environment from OpenAI gym [2]. Our implementation can be found on the dedicated GitHub repository.

Introduction

The purpose of Reinforcement Learning is to develop algorithms capable of learning the best strategy an agent should use within a defined environment. Due to the many prospects of application such as in health or robotics, Reinforcement Learning (RL) is today a hot research topic. The challenge in RL is to develop a stable, reliable, simple and efficient algorithms which could be transposed in many settings.

In this project we study a class of reinforcement learning algorithms called *Policy Gradient* methods with an emphasise on Proximal Policy Optimisation (PPO) proposed by Shulman et al.'s [13]. The standard Policy Gradient algorithms are Advantage Actor Critic [10] and REINFORCE [17]. Similarly to the TRPO algorithm [14], PPO finds a different approach and uses a surrogate loss to the *conservative policy method* introduced by Kakade et al. [6], that forms a lower bound on the performance of the policy. The corresponding maximisation problem is unconstrained and can be easily optimised with stochastic gradient ascent at a lower computational cost than TRPO. According to the authors, PPO performs comparably or better than other approaches while being much simpler to implement and tune. The original article being a rather brief and empirical paper, we aim to deepen the details of its conception and compare it to Advantage Actor Critic and TRPO. We also implement Advantage Actor Critic and PPO to compare both approaches in practice.

The mathematical framework used for Reinforcement Learning problems is the Markov Decision Process (MDP). MDPs are defined by four key components: $\mathcal{S}$ the *state space*, $\mathcal{A}$ the *action space*, $\mathbb{P}$ the *transition probability* and r the *reward* function. A simple MDP example is represented in Figure 1. At any time $t \in \mathbb{N}$, the current state of the system and the action taken are respectively represented by s_t taking value in $\mathcal{S}$ and a_t taking value in $\mathcal{A}$. It is worth noting that $\mathcal{S}$ and $\mathcal{A}$ are not necessarily countable. The transition probability gives the probability to be in a given subset of $\mathcal{S}$ given the current state, the action taken and the current time t. The reward r_t represents the gain or loss when choosing the action a_t in the current state s_t at date t. We call a *transition* a tuple formed by (s_t, a_t, s_{t+1}, r_t).

To sum up, at each time step t, the agent receives some representation of the environment's state $s_t \in \mathcal{S}$. On that basis, the agent selects an action $a_t \in \mathcal{A}$. The agent finds itself in a new step $s_{t+1} \in \mathcal{S}$ one time step later and receives a numerical reward r_t on the way. This interaction is schematised in Figure 2. We are looking for a strategy, or policy, that attributes to each state of the state space $\mathcal{S}$ a distribution over actions from the action space $\mathcal{A}$. The goal of Reinforcement Learning algorithms is to find the strategy that will maximise the cumulative sum of rewards.

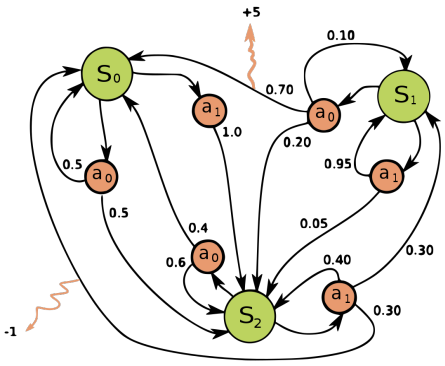


Figure 1: Markov Decision Process with three states (green circles), two actions (orange circles) and two rewards (orange arrows). Source: https://en.wikipedia.org/wiki/Markov_decision_process.

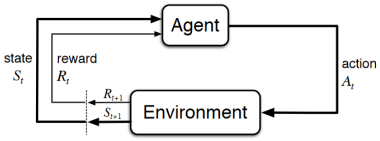


Figure 2: The agent-environment interaction in a MDP. Source: [16].

Setting First of all, we assume that the underlying environment can be modelled as a Markov Decision Process. Although some algorithms can be generalised for a Partially Observed Markov Decision Process, we stick to the simplest case. Secondly, we set ourselves in the case where each episode stops after a deterministic number of steps T. This will avoid technical considerations as we do not treat T as a random stopping times (episodic case) nor deal with the case where there can be an infinite number of steps (continuous case). Thirdly, the type of approaches we consider in this project are "model-free", meaning that the transition probabilities and reward distributions are unknown, and we do not seek to model them. Even if it seems that "model-free" can work without the MDP assumption, the theoretical convergence guarantees are only valid in that case. Finally, we consider a setting with either a continuous or a very large discrete state and/or action space. This is most suited to real-world problems. Thus, neither the value function nor the policy can be represented as vectors or tensors as in the tabular Reinforcement Learning approach. Instead, we will consider an approximate RL approach, where the value function and policy are seen as parametrised functions (usually neural networks).

Outline Our report is constructed as follows: firstly we follow the structure of Shulman's et al. article [13] in order to go deeper into the theoretical aspect of the article. We then undertake our own experiments in order to form an other view on the article. In this way, we describe first the Policy Gradient Methods 1, then the Trust Region Methods 2, including the Clipped 2.2.1, the Adaptative KL Penalty Surrogate Objectives 2.2.2 and the resulting PPO Algorithm 3. We then present our own Implementation 4 where we describe the dataset used, the hyperparameter choices and our results. Finally we explore the Limits of the article and its approach 5 before drawing a Conclusion 6 addressing areas for improvement.

1 Policy Gradient Methods

In this section we recall some important results of Policy Gradient Methods. For a good introduction to Policy Gradient Methods see Chapter 13 of Sutton and Barto [16], the course 7 of Silver [15] and lectures 5, 6 and 7 of Levine [7].

One approach in reinforcement learning consists in adopting a **Policy Iteration method**. Policy iteration methods consists in firstly estimating the value of each state (or state-action pairs) and then choosing an improved policy based on this estimation (for example with greedy or ε-greedy methods). It alternates between two phases, the *Policy Evaluation phase*; evaluating the states under the current policy, and the *Policy Improvement Phase*; improving the policy based on the estimated values. *Policy Iteration* methods only differ by how they perform the two steps. One classical example that has been successfully adapted to large scale application with deep neural networks is the Deep variant of the Q-learning algorithm: DQN [9].

Policy Gradient methods, on the other hand, consider the policy as a parametrised function π_θ. Noting $\theta \in \mathbb{R}^d$ the policy's parameter, we write $\pi_\theta(a,s) = \mathbb{P}(A_t = a | S_t = s, \theta_t = \theta)$ the probability of action a to be chosen at time t while being at state s with the parameter θ. We assume that the policy is differentiable with respect to θ. The idea of Policy Gradient Methods is to choose a performance measure $\theta \mapsto J(\theta)$ which we will try to maximise by gradient ascent : $\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$. The most natural performance measure to evaluate a policy is the average discounted returns when following the policy:

$$J(\theta) = \mathbb{E}_{\pi_\theta(\tau)}[\sum_{t=1}^{T} \gamma^t r(s_t, a_t)] = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)]$$

where $\gamma \in (0,1)$ is a discount factor, $r(.,.)$ is the reward function defined on $\mathcal{S} \times \mathcal{A}$ and taking value in $\mathbb{R}$, τ is a trajectory (i.e. a sequence of transitions) of the form $\tau = (s_1, a_1, r_1, \cdots, s_T, a_T, r_T)$ and $p_\theta(\tau)$ is the probability of the trajectory τ when the agent follows the policy π_θ, i.e.,

$$\pi_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t, s_t) p(s_{t+1} | s_t, a_t)$$

Not having access to $\nabla J(\theta_t)$, we need to estimate it. Following the *Policy Gradient Theorem* of Marbach and Tsitsiklis [8] we get,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)]$$

Directly approximating this gradient with Monte Carlo estimation will lead to high variance that we can reduce using two modifications: the *causal trick* and the *baseline trick*.

Before continuing we recall three major objects in Reinforcement learning,

- $Q^\pi(s_t, a_t)$: the q-value function, $Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[\gamma^{t'} r(s_{t'}, a_{t'}) | s_t, a_t]$, i.e. the expected total reward after taking actions a_t in a state s_t and thereafter following the policy π_θ
- $V^\pi(s_t)$: the state value function, $V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(.|s_t)}[Q^\pi(s_t, a_t)]$, i.e. the expected total reward starting from state s_t and thereafter following the policy π_θ
- $A^\pi(s_t, a_t)$: the advantage function, $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, i.e. how much extra reward is expected by taking action a_t in a state s_t and thereafter following the policy π_θ

The causality trick comes from the Markov assumption on the MDP, we notice that a policy at time t' cannot affect the reward at time t when $t < t'$. In the expectation, we can therefore truncate $R(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$. Using conditional expectation, we see that it is equivalent to replace $R(\tau)$ by the Q-value function:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta(\tau)}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t,a_t) \sum_{t'=t}^{T} \gamma^{'} r(s_{t'},a_{t'})] \\ &= \sum_{t=1}^{T} \mathbb{E}_{\pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(s_t,a_t) \mathbb{E}_{\pi_\theta(\tau)}[\sum_{t'=t}^{T} \gamma^{'} r(s_{t'},a_{t'}) | s_t, a_t] \right] \\ &= \mathbb{E}_{\pi_\theta(\tau)}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t,a_t) Q^{\pi_\theta}(s_t,a_t)]\end{aligned}$$

Secondly, we notice that we can subtract to the Q-value function any quantity $b(s)$, that can depend on the states but is independent of the actions. Indeed,

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} b(s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] &= \sum_{t=1}^{T} \mathbb{E}_{s_t \sim p_\theta(s_t)} \left[\mathbb{E}_{a_t \sim \pi_\theta(.|s_t)}[b(s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \right] \\ &= \sum_{t=1}^{T} \mathbb{E}_{s_t \sim p_\theta(s_t)} \left[b(s_t) \mathbb{E}_{a_t \sim \pi_\theta(.|s_t)} \left[\frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right] \right] \\ &= \sum_{t=1}^{T} \mathbb{E}_{s_t \sim p_\theta(s_t)} \left[b(s_t) \nabla_\theta \underbrace{\int_{\mathcal{A}} \pi_\theta(a_t|s_t) da_t}_{=1} \right] \\ &= 0\end{aligned}$$

Where $p_\theta(s_t)$ is the state distribution after running the policy π for t steps. We can therefore rewrite the gradient of the score function as,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta(\tau)}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t,a_t)(Q^{\pi_\theta}(s_t,a_t) - b(s_t))]$$

Having the freedom of choosing b, called the *baseline*, we should choose it as to minimise the variance of the estimated gradient. If we analyse the variance of the the random variable inside the expectation, we can derive that the optimal baseline is the expected return from t weighted by the gradient of the log policy (see lecture 5 of Levine [7]). In practice, just taking the expected return (without weight) is a good enough baseline. Since the expected return from s_t with respect to the policy is the Value-function this is equivalent to using the *advantage function*:

$$Q^{\pi_\theta}(s,a) - b(s) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) = A^{\pi_\theta}(s,a)$$

Leading to the gradient of the performance measure,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t,a_t) A^{\pi_\theta}(s_t,a_t)] \tag{1}$$

If we replace $A^{\pi_\theta}(s_t,a_t)$ by an estimator $\hat{A}_t$ that we assume independent of θ, we can take the gradient outside the expectation and take the primitive. We now get the main loss PG (for policy gradient) of the PPO article [13] in section 2.1:

$$L^{PG}(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} \log \pi_\theta(s_t,a_t) \hat{A}_t] \tag{2}$$

The loss L^{PG} is the classical one used in the literature for the REINFORCE algorithm [17] and the Advantage Actor Critic algorithm [10]. Both algorithm estimates the gradient in equation (1) and performs stochastic gradient ascent. REINFORCE estimates the expectation of the gradient and the advantage function with Monte Carlo while actor-critic uses a second network to estimate the advantage function (the critic). We will detail the actor-critic algorithm in later section as this is the one we are going to compare PPO against. We introduce another type of Policy Gradient algorithms, which is based on trust regions. Trust region methods include two notable methods : TRPO [14] and PPO [13]. We start with TRPO that is significantly harder to derive but reduces the problem to a quadratic optimisation problem. Furthermore, it introduces the main idea for the PPO algorithm.

2 Trust Region Policy Gradient Methods: TRPO and PPO

In the last section, we introduced a first Policy Gradient loss. The **Trust Region Methods** are based on a different formulation of the loss, derived from the *performance difference lemma*. TRPO and PPO are two examples of trust region methods. Both approaches work by finding a *surrogate* objective, which *locally* constitutes a *lower bound* on the true objective. The policy optimisation is performed through the maximisation of this surrogate objective, which will lead to a performance improvement through the maximise-minimisation principle.

Let's first compute our "true" objective. We consider as before $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$ the performance of policy π_θ. Given a current policy $\pi_{\theta_{old}}$, we want to maximise the performance of the new policy $\pi_{\theta_{new}}$ obtained after one iteration. Maximising the performance of the new policy $\pi_{\theta_{new}}$ can be written as:

$$\theta_{new} = \arg\max_\theta J(\theta) = \arg\max_\theta J(\theta) - J(\theta_{old})$$

In order to rewrite our objective, the starting point is the performance difference Lemma (see Kakade ans Langford [6]), which indicates that for any policies π, π':

$$J(\theta') - J(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta'}(\tau)}[\sum_{t=1}^{T} \gamma^t A^{\pi_\theta}(s_t, a_t)]$$

Thus, our optimisation problem becomes:

$$\theta_{new} = \arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\sum_{t=1}^{T} \gamma^t A^{\pi_{\theta_{old}}}(s_t, a_t)] \tag{3}$$

Since we only have access to $\pi_{\theta_{old}}$, the expectation cannot be computed or approximated and the expression cannot be optimised directly. We would like to replace the expectation over π_θ by an expectation over $\pi_{\theta_{old}}$, from which we can sample. The following lower bound shows that it is not absurd if π_θ and $\pi_{\theta_{old}}$ are *close* in some sense. We define D_{TV} the total variation divergence, $D_{TV}(P||Q) = \frac{1}{2}||P-Q||_1$.

$$\begin{aligned}
J(\theta) - J(\theta_{old}) &= \sum_{t=1}^{T} \mathbb{E}_{s_t\sim p_\theta(s_t)} \left[\underbrace{\mathbb{E}_{a_t\sim\pi_\theta(.|s_t)}[\gamma^t A^{\pi_{\theta_{old}}}(s_t, a_t)]}_{:=f(s_t)} \right] \\
&= \sum_{t=1}^{T} \int f(s_t) dp_\theta(s_t) \\
&= \sum_{t=1}^{T} \int f(s_t) dp_{\theta_{old}}(s_t) + \sum_{t=1}^{T} \int f(s_t) d(p_\theta - p_{\theta_{old}})(s_t) \\
&\geq \sum_{t=1}^{T} \int f(s_t) dp_{\theta_{old}}(s_t) - C \cdot D_{TV}(\pi_\theta||\pi_{\theta_{old}}) \\
&= \sum_{t=1}^{T} \mathbb{E}_{s_t\sim p_{\theta_{old}}(s_t)} \left[\mathbb{E}_{a_t\sim\pi_\theta(.|s_t)}[\gamma^t A^{\pi_{\theta_{old}}}(s_t, a_t)] \right] - C \cdot D_{TV}(\pi_\theta||\pi_{\theta_{old}})
\end{aligned} \tag{4}$$

where D_{TV} is the total variation distance and C is a positive constant that depends on the discount rate γ and $\max_s |\mathbb{E}_{a\sim\pi_\theta(.|s)}[A^{\pi_{\theta_{old}}}(s,a)]|$.

Thus, by maximising the last equation (4) in θ, we obtain a *monotonic improvement* of the policy performance. However, since π_θ is unknown, $\mathbb{E}_{a_t\sim\pi_\theta(.|s_t)}[\cdot]$ cannot be estimated. The problem is further transformed using importance sampling, in order to obtain an expectation with respect to $\pi_{\theta_{old}}$ rather than π_θ:

$$\mathbb{E}_{s_t\sim p_{\theta_{old}}(s_t)} \left[\mathbb{E}_{a_t\sim\pi_\theta(.|s_t)}[\gamma^t A^{\pi_{\theta_{old}}}(s_t, a_t)] \right] = \mathbb{E}_{s_t\sim p_{\theta_{old}}(s_t)} \left[\mathbb{E}_{a_t\sim\pi_{\theta_{old}}(.|s_t)}[\gamma^t \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A^{\pi_{\theta_{old}}}(s_t, a_t)] \right]$$

Optimising the total variation divergence D_{TV} might be computationally difficult, however the problem can be relaxed using Pinsker's inequality.[1] Given D_{KL} the Kullback Leibler divergence, $D_{KL}(P||Q) = \int_{-\infty}^{+\infty} p(x) log\left(\frac{p(x)}{q(x)}\right) dx$ where p and q are the densities of P and Q,

$$\forall\, \pi, \pi',\ D_{TV}(\pi||\pi') \leq \sqrt{\frac{1}{2} D_{KL}(\pi, \pi')} \qquad \text{Pinsker's inequality}$$

The theoretical constant C might be too conservative in practice. It can be replaced by an hyperparameter β, which leads to the following program:

$$\underset{\theta}{\text{maximise}} \sum_{t=1}^{T} \hat{\mathbb{E}}_{(s_t,a_t)\sim\pi_{\theta_{old}}} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \gamma^t \hat{A}^{\pi_{\theta_{old}}}(s_t, a_t) \right] - \beta KL[\pi_\theta(.|s_t), \pi_{\theta_{old}}(.|s_t)] \tag{5}$$

where $\hat{\mathbb{E}}_{\pi_{\theta_{old}}}$ and $\hat{A}^{\pi_{\theta_{old}}}(s_t, a_t)$ are approximations of $\mathbb{E}_{\pi_{\theta_{old}}}$ and $A^{\pi_{\theta_{old}}}(s_t, a_t)$ obtained with observations sampled from $\pi_{\theta_{old}}$. Let's note that the first part of equation (5) is a valid proxy for equation (3) if and only if the updated policy is not too far from the old one. The Kullback Leibler term enforces this requirement and all the variants we are going to see impose the same restriction. That is why they are called **Trust Region Methods**, the trust region refers to the space where the policy are close and the approximation of the performance difference lemma is acceptable.

2.1 TRPO

It's theoretically possible to estimate the gradient of equation (5) with Monte Carlo or in an actor-critic fashion and perform Stochastic Gradient Ascent. However, in practice, the parameter β is hard to tune under this 'softly' penalised form. That is why, Schulman et al. [14] replaced the soft constraint by a hard constraint. TRPO suggests solving the following constrained optimisation problem instead of equation (5):

$$\begin{aligned} &\underset{\theta}{\text{maximise}} \sum_{t=1}^{T} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \gamma^t \hat{A}_t \right] \\ &\text{subject to } \hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta \end{aligned} \tag{6}$$

Based on the result derived on equation (4), we obtain that the constrained program (6) leads to a monotonic improvement of the policy performance for a small enough δ and a sufficiently good estimation $\hat{\mathbb{E}}_t[\cdot]$. Let us now define:

$$L^{CPI}(\theta) = \sum_{t=1}^{T} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \gamma^t \hat{A}_t \right] = \sum_{t=1}^{T} \hat{\mathbb{E}}_t [r_t(\theta) \gamma^t \hat{A}_t]$$

where CPI means *Conservative Policy Iteration*, where *conservative* refers to the use of the performance difference lemma that keeps track of the previous policy.

The gradient of L^{CPI} can be computed with derivations similar to section 1. When implemented, TRPO uses a linear approximation of the objective and a quadratic approximation of the constraint. Given $F(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T\right]$ the Fisher information matrix with respect to the old parameters:

$$L^{CPI}(\theta) \approx L^{CPI}(\theta_{old}) + \nabla_\theta L^{CPI}(\theta_{old})^T (\theta - \theta_{old})$$

$$KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)] \approx \frac{1}{2} (\theta_{old} - \theta)^T F(\theta_{old}) (\theta_{old} - \theta) \tag{7}$$

The proof to derive equation (7) is simply a Taylor expansion. The final programme is therefore a standard quadratic program that can be solved efficiently using the conjugate gradient algorithm (see appendix C in [14] for details).

[1] Let's note that this trick was also used to derive lower bounds of the regret of some Bandit algorithms.

$$\underset{\theta}{\text{maximise}} \ \nabla_\theta L^{CPI}(\theta_{old})^T(\theta - \theta_{old})$$
$$\text{subject to} \ \frac{1}{2}(\theta_{old} - \theta)^T F(\theta_{old})(\theta_{old} - \theta) \leq \delta$$

The remaining difficulty if the computation of the Fisher Information matrix that prevents TRPO to scale to Deep Neural Networks for the policy.

Since the release of the TRPO algorithm in 2015 (Schulman et al. [14]), the Actor Critic using Kronecker-factored Trust Region algorithm (ACKTR, pronounced "actor") [18] lead to the the first scalable trust region natural gradient method for actor-critic method. It is based on a special approximation of the Fisher Information Matrix called *Kronecker-factored approximation*. Both PPO and ACKTR were released in 2017, therefore the articles do not contain a comparison of their performances.

2.2 PPO

2.2.1 Clipped Surrogate Objective

As mentioned before, the use of the first term in equation (5) is only justified if we control the deviation between the old policy and the new one. Directly maximising the objective without controlling the deviation leads to excessively large policy updates if not penalised or constrained. We can either use soft constraint of hard constraint like the TRPO objective. As opposed to the TRPO approach, the authors of the PPO algorithm [13] suggest a surrogate objective which can be directly used in an unconstrained program. This objective, called clipped surrogate objective, is a lower bound of the initial objective L^{CPI}. As in the case of TRPO, the local optimisation of this surrogate objective *locally* leads to a monotonic performance improvement, based on the maximise-minimisation principle.

Let $r_t(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ be the probability of taking action a_t in state s_t following the new policy divided by the same probability for the old policy as in the last section. Let's define $\varepsilon > 0$, the hyperparameter used to control the variation of the probability ratio $r_t(\theta)$. The smaller ε, the smaller the policy steps will be, and conversely. Then, the clipped surrogate objective L^{CLIP} is defined as:

$$\begin{aligned} L^{CLIP}(\theta) &= \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \\ &= \begin{cases} \hat{\mathbb{E}}_t[\min\left(r_t(\theta), 1+\varepsilon\right)\hat{A}_t] & \text{if } \hat{A}_t \geq 0 \\ \hat{\mathbb{E}}_t[\max\left(r_t(\theta), 1-\varepsilon\right)\hat{A}_t] & \text{if } \hat{A}_t \leq 0 \end{cases} \end{aligned} \tag{8}$$

By definition, L^{CLIP} is a lower bound of L^{CPI}. It can indeed be noted from equation (8) that the probability ratio is clipped only when it leads to an improvement of the objective. For instance, if the estimated advantage $\hat{A}_t$ following the old policy is positive, the probability ratio will only be clipped if it is higher than $1+\varepsilon$, so that "good" actions are not too much favoured. In particular, the ratio will not be clipped if the probability of taking the action a_t decreases when following the policy π_θ (in comparison to $\pi_{\theta_{old}}$), although it leads to a positive advantage. Clipping the ratio in this case would lead to an overestimation of the surrogate objective, which could be higher than our true objective equation (2). As a consequence, the monotonic improvement property of the maximise-minimisation approach would not necessarily hold. Thus, as mentioned by the authors, this surrogate objective can be seen as a "pessimistic" bound that penalises for having too large of a policy update.

2.2.2 Adaptative KL Surrogate objective

The author also suggests an alternative to the clipped surrogate objective. The difficulty in the soft version of TRPO is to find the appropriate penalisation constant. To solve this problem they suggest an *adaptive penalty coefficient* $\beta^{adaptative}$ to the KL divergence. If a large update is performed (measured by the KL divergence) the constant will increase for the next iteration such that large steps are more penalized. On the contrary, if the update is too small, the constant decreases for the next iteration. The adaptative KL objective is defined as follows:

$$L^{KLPEN}(\theta) = \sum_{t=1}^{T} \hat{\mathbb{E}}_{s_t \sim p_{\theta_{old}}(s_t)} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \gamma^t \hat{A}^{\pi^{old}}(s_t, a_t) \right] - \beta^{adaptive} KL[\pi_\theta(.|s_t), \pi_{\theta_{old}}(.|s_t)] \quad (9)$$

Compute $d = KL[\pi_\theta(.|s_t), \pi_{\theta_{old}}(.|s_t)]$

- if $d < d_{targ}/1.5, \quad \beta \leftarrow \beta/2$
- if $d > d_{targ} \times 1.5, \quad \beta \leftarrow \beta \times 2$

Empirically, the authors found that the clipped surrogate objective performs better.

3 Algorithms

In the numerical experiments section, we will compare a Policy Gradient algorithm as in section 1 against the PPO algorithm with the KL loss and the clipped loss. In this section, we give the details and pseudo-code of both algorithms. We omit the REINFORCE and the TRPO algorithms as comparing more than two algorithms is too time-consuming. Each algorithm requires a not so simple implementation and especially heavy fine-tuning as their are extremely unstable.

3.1 Policy Gradient Method: the Advantage Actor-Critic (A2C) example

We recall that the gradient we seek to approximate when doing Policy Gradient updates as in section 1 has the following form:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[A^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(s_t, a_t)] \quad (10)$$

The REINFORCE[2] algorithms introduced in the seminal article of Williams [17] simply uses Monte Carlo estimation with samples from policy π_θ to estimate equation (10). It gives an unbiased estimate but with high variance. These MC estimates work poorly in practice and that is why we need the Actor-Critic variant.

Actor-Critic, unlike REINFORCE belongs to the *bootstrapping* family of algorithms. It introduced a second network $v_w(s)$ parametrised by w that will learn the value-function $V^{\pi_\theta}(s)$. In this framework we refer to π_θ as the **actor** and to v_w as the **critic**. Recall that $A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$. But we have said that the critic approximates the value function and not the advantage function, so how do we proceed ? Let us define,

$$\begin{aligned} \delta_{t,n}^{\pi_\theta} &:= r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \ldots + \gamma^{n-1} r(s_{t+n-1}, a_{t+n-1}) + \gamma^n V^{\pi_\theta}(s_{t+n}) \\ \hat{A}_t^n &:= \delta_{t,n}^{\pi_\theta} - V^{\pi_\theta}(s_t) \end{aligned}$$

$\delta_{t,n}^{\pi_\theta}$ is called the *Temporal Difference* (TD-) target. If we reach the end of the episode before n, we truncate the sum, and if s_{t+n} is a terminal state we set $V^{\pi_\theta}(s_{t+n}) = 0$. $\hat{A}_t^n$ is an unbiased estimator of $A^{\pi_\theta}(s,a)$, indeed,

$$\begin{aligned} \mathbb{E}[\hat{A}_t^n | s_t, a_t] &= \mathbb{E}[\sum_{k=0}^{n-1} \gamma^k r(s_{t+k}, a_{t+k}) + \gamma^n V^{\pi_\theta}(s_{t+n}) | s_t, a_t] - V^{\pi_\theta}(s_t) \\ &= Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \quad \text{from the Bellman equation} \\ &= A^{\pi_\theta}(s_t, a_t) \end{aligned}$$

If we have access to the true value function we can use $\hat{A}_t^n$ as an estimator for the advantage function. However, not having access to V^{π_θ} we plug instead the critic network v_w:

[2] Reward Increment = Non negative Factor times Offset Reinforcement times Characteristic Eligibility

$$\delta^w_{t,n} := r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \ldots + \gamma^{n-1} r(s_{t+n-1}, a_{t+n-1}) + \gamma^n v_w(s_{t+n})$$
$$\hat{A}^n_{t,w} := \delta^w_{t,n} - v_w(s_t)$$

Since v_w is usually not perfectly trained, it will introduce a bias in the advantage estimation, but it can be shown that the variance is reduced. In fact, if n goes to infinity we recover the Monte Carlo setting with no bias and high variance, and if $n = 1$ we get an estimator with a low variance but high bias. There is therefore a bias-variance trade-off induced by n, and introducing a bit of bias but reducing the variance has often a huge positive impact on the performances. n is usually treated as an hyper-parameter and chosen by evaluating the critic performances after different training.

We therefore approximate equation (10) with our critic estimation,

$$\nabla_\theta J(\theta) \approx \hat{A}^n_{t,w} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

where $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is typically computed using a library that handles automatic differentiation like PyTorch [11]. We then use Stochastic Gradient ascent.

The last question remained to answer is: how to train the critic ? Until now, we have assumed that v_w was already trained. The critic takes as input a state s_t and outputs an estimate of the state value $v_w(s_t)$. In fact, the target used to train the critic are the TD target (hence the name) $\delta^w_{t,n}$ introduced before. It is not a supervised training nor an unsupervised training as we have targets but the targets include estimationd from the network. That is why we called it a bootstrap method, we train the network with targets build from previous estimates. The mean squared error is used as a loss between the targets and the output of the critic:

$$L^{VF}_t(w) = \frac{1}{2}\left(v_w(s_t) - \delta^w_{t,n}\right)^2 = (\hat{A}^n_{t,w})^2$$

Since the targets are considered as the ground truth for training we do not differentiate them when we take the gradient of the loss,

$$\nabla_w L^{VF}_t(w) = \left(v_w(s_t) - \delta^w_{t,n}\right) \nabla_w v_w(s_t) = \hat{A}^n_{t,w} \nabla_w v_w(s_t)$$

This is a source of instability and many many tricks exist in the literature to stabilise the process like the use of a replay memory buffer or two copy of the same network that are periodically updated. It makes sense to minimise this loss as if we reach zero, v_w will satisfy the Bellman equation and we know that V^{π_θ} is the unique fixed point of this equation, therefore $V^{\pi_\theta} = v_w$. For more details about the use of Bellman equations to learn value function see [16] or any good book on Dynamic Programming.

In practice, we sample n transitions, we build the advantages estimates, we build the losses for all transitions between 1 and n and perform a single update with the aggregated gradients. The pseudo-code for an Advantage Actor-Critic Algorithm is given in 3.1.

Advantage Actor-Critic

Initialise θ the actor weights and w the critic weights, $t \leftarrow 1$
repeat
reset gradients, $d\theta \leftarrow 0$, $dw \leftarrow 0$
$t_{start} = t$, get s_t
repeat
Perform $a_t \sim \pi_\theta(s_t, .)$ and observe (r_t, s_{t+1})
$t \leftarrow t+1$
until s_t is terminal or $t - t_{start} = n$
if s_t is terminal $R = 0$, otherwise $R = v_w(s_t)$
for $i \in \{t-1, \ldots, t_{start}\}$,
$R \leftarrow r_i + \gamma R$
accumulate gradients $d\theta \leftarrow d\theta + \alpha(R - v_w(s_i))\nabla_\theta \log \pi_\theta(s_i, a_i)$
accumulate gradients $dw \leftarrow dw + \beta(v_w(s_i) - R)\nabla_w v_w(s_i)$
end for
perform one gradient update with $d\theta$ and dw
until max steps or satisfying scores is reached

Several remarks can be made on this pseudo-code. First of all, there exists a lot of tricks to improve this basic code. For instance, reward clipping and normalisation, mini-batch updates, replay memory buffers to re-sample old transitions, or shared weights between the actor and the critic can be introduced. In practice the actor and the critic are neural networks, which require additional hyper-parameters. The fine tuning of A2C's parameter is delicate as it is not really stable. Secondly, we can see that using bootstrap, if $n < T$, we do not have to wait the end of the episode before updating the networks. It can therefore also handle continuous task where an episode never end. This property is not satisfied by the REINFORCE algorithm. Finally, there exists some variants that run multiple agent in parallel and lead to great results, for instance in the A3C [10] framework.

3.2 Trust Region Policy Gradient Method: the PPO example

For the PPO algorithm we will only detail the case of the clipped surrogate loss, as it is the most efficient in practice. Recall that we want to estimate,

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

The training process is similar to the Advantage Actor Critic training loop. Their are two main differences (on top of the "raw" loss differences). Firstly, instead of using n-step TD targets to build $\hat{A}_t$ as in the last section they use a new way of estimating advantage functions: the Generalised Advantage estimation [12]. Secondly, they also add an entropy penalisation to enforce better exploration. The critic, however, is trained exactly as for A2C.

3.2.1 Generalised Advantage Estimation.

Recall that in the last section we defined,

$$\begin{aligned} \delta_{t,n}^{\pi_\theta} &:= r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \ldots + \gamma^{n-1} r(s_{t+n-1}, a_{t+n-1}) + \gamma^n V^{\pi_\theta}(s_{t+n}) \\ \hat{A}_t^n &:= \delta_{t,n}^{\pi_\theta} - V^{\pi_\theta}(s_t) \end{aligned}$$

where n is an hyper-parameter to tune. With Generalised Advantage Estimation (GAE) we want to use $\hat{A}_t^n$ for all n and average them to get a more powerful estimator. Let's consider en exponential weighted average with parameter $\lambda \in [0, 1]$.

$$
\begin{aligned}
\hat{A}_t^{GAE} &= (1-\lambda)\sum_{n\geq 1}\lambda^{n-1}\hat{A}_t^n \\
&= (1-\lambda)\sum_{n\geq 1}\lambda^{n-1}\left(\sum_{k=0}^{n-1}\gamma^k\left(\delta_{t+k,1}^{\pi_\theta} - V^{\pi_\theta}(s_{t+k})\right)\right) \\
&= (1-\lambda)\sum_{k\geq 0}\left(\gamma^k\left(\delta_{t+k,1}^{\pi_\theta} - V^{\pi_\theta}(s_{t+k})\right)\sum_{n\geq k+1}\lambda^{n-1}\right) \\
&= \sum_{k\geq 0}(\lambda\gamma)^k\left(\delta_{t+k,1}^{\pi_\theta} - V^{\pi_\theta}(s_{t+k})\right)
\end{aligned}
$$

Therefore by taking a weighted average of every n-step TD target we obtain a simple expression that only depends on λ. The parameter λ plays an equivalent role as n in the last section. If $\lambda = 1$, we recover the Monte Carlo setting with no bias and high variance, and if $\lambda = 0$ we get an estimator with a low variance but high bias. [12] showed that this estimator have better performance than n-step TD estimates. In practice, we truncate the sum at $k = T$. The resulting estimator, used in PPO, is called Truncated Generalised Advantage Estimator.

3.2.2 Entropy for exploration

The entropy of a random variable can be interpreted as the average level of *surprise*, or *uncertainty* inherent in the variable's possible outcomes. Therefore the highest the entropy of the actor policy the more it will explore. The entropy is thus a natural way to enforce exploration.

In the discrete case, we define the entropy as:

$$S[\pi_\theta](s_t) = -\sum_{a\in\mathcal{A}}\pi_\theta(a|s_t)log(\pi_\theta(a|s_t))$$

If a policy is deterministic, it will attribute a probability 1 to only one action of the action space, and the entropy will be zero. On the opposite, the entropy is maximal if the actor attributes uniform probabilities to each action. In the continuous case, the entropy is defined the following way:

$$S[\pi_\theta](s_t) = -\int_{\mathcal{A}}\pi_\theta(a|s_t)log(\pi_\theta(a|s_t))\,da$$

In practice, it is common to use a Gaussian distribution $\mathcal{N}(\mu_t, \sigma_t)$ for the policy in the continuous case. The entropy is then,

$$S[\pi_\theta](s_t) = -\frac{1}{2}log(2\pi\sigma_t^2) + 1$$

Therefore, if the standard deviation is constant along states (i.e. $\sigma_t \equiv \sigma\ \forall\, t$), the entropy bonus will stay constant and is useless. This is the case if the actor network only outputs the mean μ_t of the Gaussian distribution and the standard deviations are fixed. In this case, the standard deviation is set constant and manually fine tuned.

Moreover, including an entropy bonus enables to make the policy "more stochastic", and thus facilitate the optimisation. Indeed, according to the experiments of [1], even if the gradient is known exactly, gradient ascent without entropy bonus leads to suboptimal policies 25% of times due to the non-concave and unsmooth nature of the objective landscape. Including an entropy bonus could make the objective landscape easier to optimise, and thus serve as a sort of regulariser on top of encouraging exploration.

3.2.3 Global objective

All together, our final objective function contains two parts: one for the critic, which is optimised with respect to the parameter w, and one for the actor network, which depends on θ:

$$
\begin{cases}
\hat{\mathbb{E}}_t\left[L_t^{VF}(w)\right] &= \hat{\mathbb{E}}_t\left[\left(v_w(s_t) - v_t^{targ}\right)^2\right] \qquad \text{(critic with parameter } w\text{)} \\
\hat{\mathbb{E}}_t\left[L_t^{Actor}(\theta)\right] &= \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) + c_2 S[\pi_\theta](s_t)\right] \qquad \text{(actor with parameter } \theta\text{)}
\end{cases} \tag{11}
$$

where :

- $\hat{\mathbb{E}}_t[\cdot]$ is the empirical average over a finite batch of samples
- L_t^{VF} is the critic *loss* exactly like for the A2C algorithm
- L_t^{CLIP} is the clipped *objective* used to optimise the actor network
- $S[\pi_\theta]$ is the entropy regularisation
- c_2 is a manually tuned positive constant
- L_t^{Actor} is the global objective of the actor network

The actor objective and the critic loss of equation (11) can be rewritten as a single global objective, which is optimised with respect to θ and w :

$$L_t(\theta, w) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - L_t^{VF}(w) + c_2 S[\pi_\theta](s_t) \right] \tag{12}$$

In particular, if allow parameter sharing between the actor and the critic networks (therefore θ represent both the critic and the actor weights), one can introduce the positive hyper-parameter c_1 to balance between the actor objective and the critic loss, and redefine the global objective as :

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

The pseudo-code for the PPO algorithm without parameter sharing between the actor and the critic and for the clipped loss is presented below. The advantage are estimated using the GAE with parameter λ and the critic is updated with the n-step TD targets. On the contrary to A2C, we reuse each samples several times with K epochs.

PPO with clipped loss

Initialise θ the actor weights and w the critic weights, $t \leftarrow 1$
repeat
  reset gradients, $d\theta \leftarrow 0$, $dw \leftarrow 0$
  $t_{start} = t$, get s_t
  repeat
    Perform $a_t \sim \pi_{\theta_t}(s_t, .)$ and observe (r_t, s_{t+1})
    $t \leftarrow t + 1$
  until s_t is terminal or $t - t_{start} = n$
  if s_t is terminal $V_t = 0$, otherwise $V_t = v_w(s_t)$
  $R \leftarrow V_t$
  $\delta \leftarrow 0$
  for $i \in \{t-1, \ldots, t_{start}\}$,
    $R \leftarrow r_i + \gamma R$
    $\delta \leftarrow R - v_w(s_i)$
    $A \leftarrow A + \lambda\gamma\delta$
    accumulate gradients $d\theta \leftarrow d\theta + \alpha \nabla_\theta L_i^{CLIP}(\theta)$, where,
    $\nabla_\theta L_i^{CLIP}(\theta) =$
    $\begin{cases} \nabla_\theta L_i^{CPI}(\theta) & \text{if } \frac{\pi_\theta(a_i|s_i)}{\pi_{old}(a_i|s_i)} \in [1 \pm \varepsilon] \text{ or } L_i^{CLIP}(\theta) < L_i^{CPI}(\theta) \\ 0 & \text{otherwise} \end{cases}$
    $\nabla_\theta L_i^{CPI}(\theta) = \frac{A}{\pi_{old}(a_i|s_i)} \nabla_\theta \pi_\theta(a_i|s_i)$
    accumulate gradient $dw \leftarrow dw + \beta(v_w(s_i) - R)\nabla_w v_w(s_i)$
  update θ and w with $d\theta$ and dw for K epochs
until max steps or satisfying scores is reached

3.3 Discrete versus Continuous Action Space

One serious advantage of Policy Gradient methods over Policy Iteration methods such as Q-learning is that it can handle a continuous action space. If the action space $\mathcal{A}$ is discrete, we make the neural network of the actor outputs $|\mathcal{A}|$ values: $(o(s)_1, \dots, o^(s)_{|\mathcal{A}|})$ and we define a categorical distribution with a softmax normalisation:

$$\pi_\theta(s,a) = \frac{e^{o(s)_a}}{\sum_{a'=1}^{|\mathcal{A}|} e^{o(s)_{a'}}} \qquad \forall a \in \{1, \dots, |\mathcal{A}|\}$$

If $\mathcal{A}$ is continuous we can select actions according to a Gaussian distribution and make the actor network outputs parametrise this distribution. Let's assume that $\mathcal{A} = \mathbb{R}$. The actor network will output $(\mu(s), \rho(s))$. To get a positive scaling parameter we make the transformation $\sigma^2(s) = \log(1 + \exp(\rho(s))$. We construct the policy with:

$$\pi_\theta(s,.) \sim \mathcal{N}(\mu(s), \sigma^2(s))$$

Multidimensional extensions of those cases with multivariate categorical distributions and multivariate Gaussian distributions (or any continuous parametric distribution) can also be considered, it simply increases the number of outputs as we can parametrise the dependencies between the dimensions of the actions.

4 Numerical Experiments

We implemented the PPO and Advantage Actor Critic algorithm from scratch with PyTorch [11]. We build one unique class, which can be used with the clipped surrogate and the adaptative KL surrogate for PPO, or the advantage actor critic objective. Our code can be trained both on discrete and continuous control tasks and domains. Thus, it can deal both with discrete and continuous action and state spaces. However, we experimented difficulties with continuous action spaces as it requires much more time and fine tuning to make it work. In this section, we present an example with a discrete action space.

Environment We use the CartPole environment of OpenAI Gym [2]. A pole is attached to a cart, which moves along a track (without frictions). When the environment is reset, the pendulum starts upright. The goal is to prevent the cart from falling over by increasing and reducing the cart's velocity. The episode terminates whenever the cart falls, which is a random stopping time. Thus, this environment corresponds to an episodic setting. The state space has four dimensions, which are continuous: the cart position, the cart velocity, the pole angle and the pole velocity at tip. The action space contains two discrete values, which correspond to pushing the cart to the left or to the right. A screenshot of the CartPole environment is displayed Figure 3.

4.1 Networks' architecture

In our implementation, we optimise separately the actor and the critic networks, by minimising the opposite of the global objective defined in equation (12). We use the same feed-forward architecture for the actor and the critic as in the article. For both networks, we build simple feed-forward networks with two hidden layers of size 64, only the output layer size changes: one output for the critic and an output layer with the same size as the action space for the actor. At first, we used hyperbolic tangent activation functions, but when dealing with the continuous case we had a vanishing gradient phenomenon that disappeared when we used ReLU activation functions instead. For the discrete case, we add a softmax layer in order to obtain a distribution over actions. In case of continuous control problems, the network outputs the mean of a Gaussian distribution for each action dimension. We manually tune the standard deviation, and we assume that the different dimensions of the action space are independent, which yields a diagonal covariance matrix (though this assumption could be relaxed).

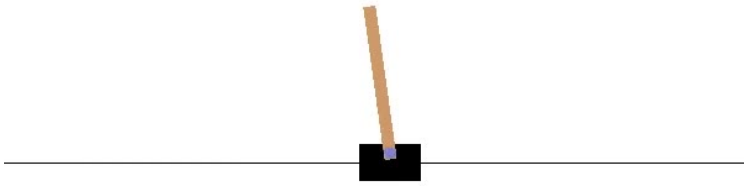


Figure 3: A screenshot of the CartPole environment. The pole is falling to the left, therefore the agent should push the cart to the left to balance it.

4.2 Results and Sensitivity analysis

From our tests, we found PPO (both with clipped and adaptative KL objectives) to be quite sensitive to fine tuning and optimisation configuration, and so as A2C in a lesser extent. After a minimum fine-tuning, based on some best practices for the PPO algorithm[3], we trained each algorithm with their respective best hyper-parameters on 1200 games, we obtain the performance presented Figure 4. We obtain quite satisfactory results for the clipped loss that reach more than 400 points in average while the A2C plays perfectly at the end, the maximum possible score on CartPole being 500. However, this is not the case for the adaptive KL objective, where the performance roughly reaches 100. After training we evaluate them on 100 games, the summary statistics are presented in Table 1.

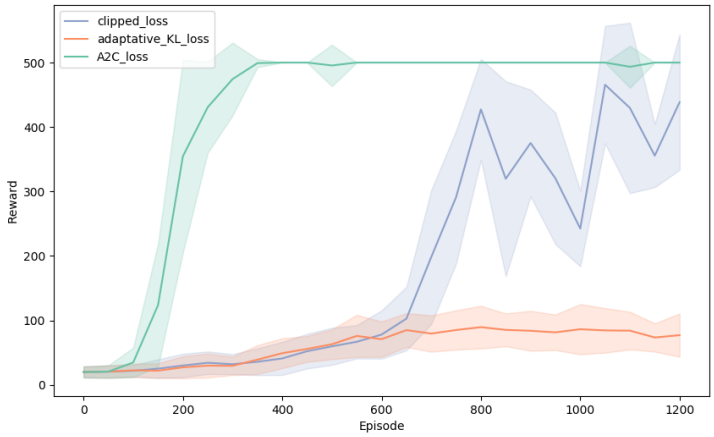


Figure 4: Evolution of the sum of rewards on Cart Pole v1 for A2C and PPO with the clipped and adaptative KL loss.

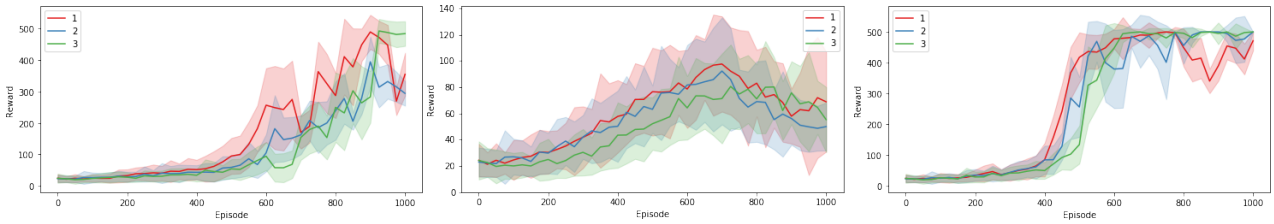
In order to check the robustness of the implemented approaches, we perform several verifications and sensitivity analysis. We first check that the results are consistent when changing the random

[3]See for instance https://github.com/llSourcell/Unity_ML_Agents/blob/master/docs/best-practices-ppo.md

	Clipped (PPO)	Adaptative KL (PPO)	A2C
mean	326	78	496
std	163	26	39
min	53	42	101
25%	167	60	500
50%	364	74	500
75%	500	87	500
max	500	171	500

Table 1: Summary statistics of the three algorithms on an evaluation test on 100 games.

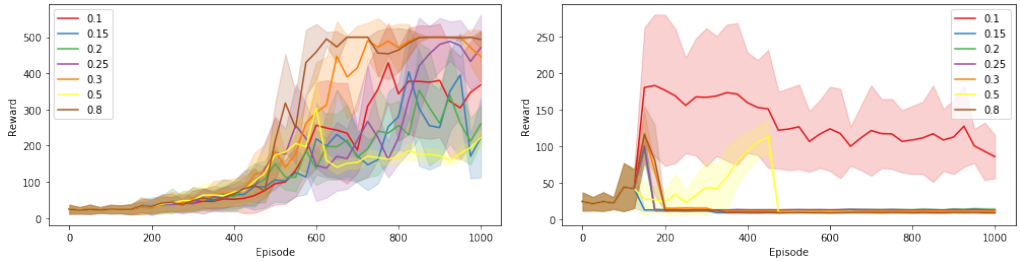
seed. The results are presented Figure 5. The trends are consistent for the three objectives, although their impact on the performance of the algorithm is not negligible, as often in RL.



(a) Clipped (b) Adaptative KL (c) A2C

Figure 5: Performance with the Clipped, Adaptative KL and A2C objectives, *for three different random seeds*

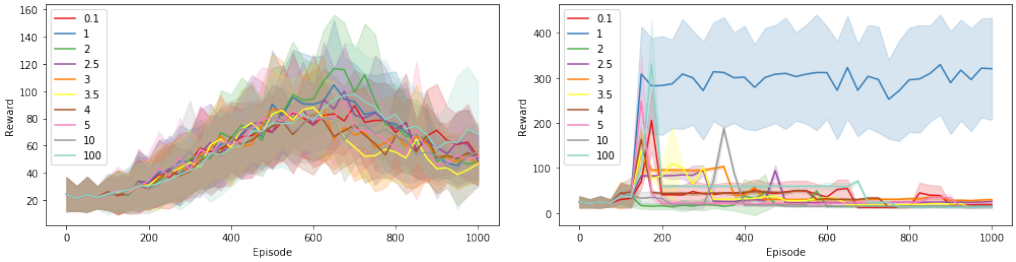
Figure 6 represents the performance of PPO with the clipped objective, for different values of the clip parameter ε, where we use either $K = 4$ or $K = 1000$ epochs. We can see than the performance is very sensitive to the value of ε. Moreover, the optimal value of ε highly varies depending on the other hyperparameters. In particular, for a given value of ε, the shape of the performance curve is very different whether the model has been trained for a few epochs (Figure 6a), or many epochs (Figure 6b). Intuitively, if the model is trained for a very large number of epochs, ε should be set to a small value in order to avoid a very large update of the policy. With 1000 epochs, the best results are obtained with the lowest value, $\varepsilon = 0.1$, which confirms this intuition.



(a) 4 epochs (b) 1000 epochs

Figure 6: Sensitivity analysis with the Clipped objective for different values of ε

Regarding the adaptative KL objective, an analysis of the sensitivity to the initialisation of the coefficient $\beta^{adaptative}$ are presented Figure 7. As a reminder, the authors claim that the algorithm is not very sensitive to $\beta^{adaptative}$, and advise to use a value of 1. Indeed, we find that in case of a few epochs (Figure 7a), the results are rather similar for different values of $\beta^{adaptative}$. However, it is not the case if we use more epochs. For 1000 epochs, we find that the optimal value is $\beta^{adaptative} = 1$, as recommended by the authors. However, we cannot conclude that the algorithm is insensitive to $\beta^{adaptative}$, and this parameter could require some fine tuning in practice.



(a) 4 epochs

(b) 1000 epochs

Figure 7: Sensitivity analysis with the Adaptative KL objective for different values of $\beta^{adaptative}$

In both Figures 6 and 7, we can observe that the performance of the policy tends to decrease after a certain number of episodes for some values of ε and $\beta^{adaptative}$. This could be due to an overfitting problem, which could be solved by setting a proper stopping criteria. OpenAI Gym includes information about the "solved returns" of each environment. In particular, we can used this value as a stopping criterion (500 in the case of Cartpole). However, when policies do not manage to reach this threshold, it is possible to use another metric, for instance the evolution of the average returns over 100 episodes. Finally, in the Cartpole environment, 4 epochs do not seem to be enough for the adaptative KL objective, although it is enough for the A2C objective. On the opposite, 1000 epochs most probably involve some overfitting.

From our implementation, we can conclude that the PPO algorithm requires some custom and thorough fine tuning in order to perform properly. However, we implemented a "minimal" version of the algorithm, and other extensions could be added.

4.3 Extensions

Gaussian actor network In our implementation, the mean of the Gaussian distribution used to sample actions from the policy is outputted by the actor network. However, the covariance matrix is manually fine tuned, and the different dimensions are assumed to be independent. Our network could be easily modified so as to also output both the mean and the covariance matrix of the distribution. Moreover, the covariance matrix does not need to be diagonal, thus we could easily relax the assumption of independence between action dimensions.

Decreasing entropy bonus coefficient As designed in the PPO article, we used a constant coefficient c_2 to balance the entropy bonus. However, once the policy has improved, the need for exploration is reduced, and the agent can start exploiting more. Thus, one could think of designing a sequence of (strictly positive) coefficients that would decrease along time (though it would imply more fine tuning). This is a common process in Reinforcement Learning, like the decay of ε when using $\varepsilon-$greedy exploration.

Bounded continuous control tasks For continuous control tasks, the authors suggest building an actor network that outputs the mean of a Gaussian distribution. However, the action space may be bounded (on top of being continuous). This is the case in many real life settings (which can include "physical" constraints) or OpenAI Gym settings. If the bounds are "soft", the environment may just add a penalty to the rewards in order to encourage actions within the bounds, and a Gaussian can be used as usual. However, when the bounds are strict, actions either need to be clipped or sampled from a different distribution. If actions are clipped, [5] suggests a different policy gradient estimator with reduced variance, based on the knowledge of the actions that are clipped. Other possibilities would be to directly use distributions with bounded supports. In particular, one could consider a truncated Gaussian distribution or a beta distribution, which yields better results than action clipping according to the experiments of [3].

Benchmark Finally, it would have been interesting to benchmark the two PPO objectives against other methods, aside from A2C. PPO enables the use of dropout, and parameter sharing between actor and critic, which is not the case of TRPO. In particular, the latter could help learn common

features specific to the game considered. As a naive benchmark, one could have also added a random agent.

5 Limits

From the author's benchmark, PPO seems to be relatively sample efficient *empirically*, at least with respect to other policy gradient methods, i.e it requires less samples than other methods to reach a given accuracy. However there is no evidence of this fact in the experiments and it can be noted that the number of required time steps before reaching a good performance is still in the order of one or several dozen million.

When compared to other methods, the algorithm seems to require much less fine tuning. According to the author, the only specific parameters to tune are the clipping parameter ε or the KL-divergence target of the adaptative KL objective d^{targ}. In particular, they argue that the constants involved in the update of the parameter $\beta^{adaptative}$ and its initial value $\beta_0^{adaptative}$ do not impact the performances of the algorithm, although no sensitivity analysis is provided.

Moreover, as analysed in [4], PPO baselines include some code level optimisations whose impact on the performance of the algorithm seems far from negligible. In particular, they list 9 types of optimisations, that either modify the environment (reward scaling and clipping, observation normalising and clipping) or directly the learning process (Adam learning rate annealing, orthogonal initialisation of the layer weights and layer scaling, use of hyperbolic tangent activations, global gradient clipping), and even the global objective (value function clipping). None of those tricks are mentioned in the paper and can only be found in the core code. For instance, they do not mention that in practice they use the following value function mean squared error loss,

$$L_t^{VF}(\theta) = min\left\{\left(V_\theta(s_t) - V^{targ}\right)^2, \left(clip(V_{\theta_{old}}(s_t) - \varepsilon, V_{\theta_{old}}(s_t) + \varepsilon) - V^{targ}\right)^2\right\}$$

instead of $L_t^{VF}(\theta) = \left(V_\theta(s_t) - V^{targ}\right)^2$. After an ablation study on 4 of these modifications, the authors in [4] find an important contribution of these modifications to PPO's performance. They argue that the most of the performance upgrade of PPO with respect to TRPO are in fact due to the absence of such code level optimisations in the initial TRPO baseline. After including them in TRPO ("TRPO+") or removing them from PPO ("PPO-Minimal"), they conclude that the introduction (or not) of these modifications seems to have a bigger impact on the performance than the choice of the algorithm itself. In their experiment on OpenAI Gym's Hopper environment, they measure that roughly 21% of TRPO's performance and 17% of PPO's could be due to these optimisations techniques.

On top of comparing PPO with code level optimisations and "PPO-minimal", the authors also analyse the performance of "PPO-NoCLIP", for which they *remove the ratio clipping* in the loss function, but keep the code level optimisations. On three OpenAI Gym tasks, they find uniformly better results, compared to "PPO-minimal", which does include clipping but no special code-level optimisations. In some settings, the results of PPO-NoCLIP can even match the performance of the standard PPO algorithm, that both includes clipping and code-level optimisations. These results tend to suggest that the introduction of code-level optimisation would have an bigger impact on PPO's performance than the clipping mechanism in the loss function.

According to the author, in the case of PPO, this high dependence could be due to the violation of the trust region. At the first step, the "old" policy is not available, and is thus always initialised to the current policy (to avoid clipping). Thus, the first step induces a maximisation over the *unclipped* objective, and thus potentially a violation of the trust region depending the objective's steepness. This could explain the dependency on the optimisation method rather than on the objective itself.

To briefly sum-up, although PPO seems to require less fine-tuning than other methods, its performance is still highly dependent on many tricks and code-level optimisations.

6 Conclusion

Proximal Policy Optimization is a relatively robust and flexible class of Reinforcement Learning algorithms that achieves state of the art results on several tasks. It is based on a model-free and on-policy learning framework, which tries to optimise directly a parametrised stochastic policy. Our work has provided an insight into the intuitions behind the design of the algorithm. On the way we investigate in depth TRPO and A2C as a comparison. The main contribution of the article consists in a new surrogate objective, which enables to solve an unconstrained optimisation problem in order to update the policy. Under some conditions, this framework yields a monotonic improvement in policy performance, as TRPO before.

The algorithm seems robust to different settings, both continuous and discrete, and allow many customisations. It is relatively sample efficient, with respect to other policy gradient methods at least. Nonetheless, an important number of steps still need to be performed before reaching an acceptable performance. Finally, the authors claim that PPO requires much less fine tuning than other methods. However, from our experiments and other reviews, we can conclude that PPO is still very sensitive to fine tuning and code level optimisations.

Our implementation can be used either with the clipped, adaptative KL or A2C objective, both for continuous and discrete control tasks and domains. On a quite simple environment, we manage to reach the "solved returns" with the A2C and clipped objectives, but not the adaptative KL objective. However, we would most probably obtain better results with a more thorough fine tuning, and the introduction of the code-level optimisation tricks included in PPO's baselines.

References

[1] Zafarali Ahmed et al. "Understanding the Impact of Entropy on Policy Optimization". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019, pp. 151–160.

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[3] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. "Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 834–843. URL: http://proceedings.mlr.press/v70/chou17a.html.

[4] Logan Engstrom et al. "Implementation Matters in Deep RL: A Case Study on PPO and TRPO". In: *International Conference on Learning Representations*. 2020.

[5] Yasuhiro Fujita and Shin-ichi Maeda. "Clipped Action Policy Gradient". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1597–1606. URL: http://proceedings.mlr.press/v80/fujita18a.html.

[6] Sham Kakade and John Langford. "Approximately Optimal Approximate Reinforcement Learning". In: *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*. Ed. by Claude Sammut and Achim Hoffman. San Francisco, CA, USA: Morgan Kauffman, 2002, pp. 267–274. ISBN: 1-55860-873-7. URL: http://ttic.uchicago.edu/~sham/papers/rl/aoarl.pdf.

[7] S Levine. *Deep Reinforcement Learning, CS 285 at UC Berkeley*. 2018. URL: http://rail.eecs.berkeley.edu/deeprlcourse/.

[8] P. Marbach and J.n. Tsitsiklis. "Simulation-based optimization of Markov reward processes". In: *IEEE Transactions on Automatic Control* 46.2 (2001), pp. 191–209. DOI: 10.1109/9.905687.

[9] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[10] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[11] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[12] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[13] John Schulman et al. "Proximal Policy Optimization Algorithms." In: *CoRR* abs/1707.06347 (2017). URL: https://arxiv.org/pdf/1707.06347.pdf?fbclid=IwAR3Mw3xr0xZKKi0qOohA-sz8MMlZfZ1zAuI6cD5TE622hsbCcHVSoolv-04.

[14] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1889–1897. URL: http://proceedings.mlr.press/v37/schulman15.html.

[15] D Silver. *UCL Course on RL*. 2015. URL: https://www.davidsilver.uk/teaching/.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[17] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[18] Yuhuai Wu et al. "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation". In: *Advances in neural information processing systems*. 2017, pp. 5279–5288.