Reinforcement Learning Project

PROXIMAL POLICY OPTIMIZATION – PPO

Mountain Car Continuous – Humanoid

TensorFlow 2 - Python 3

Fabian Humberto Fonseca Aponte | 1886565 | Machine Learning

MSc. Artificial Intelligence and Robotics

Sapienza Università di Roma

03/02/2020

1. Problem statement

In the present report, is discussed the implementation of the reinforcement learning algorithm Proximal Policy Optimization PPO[1] in the framework TensorFlow using the language program Python in its third version. To evaluate the performance of the implementation were used two environments of the OpenAI Gym toolkit along with the MuJoCo dynamics which are Mountain Car in continuous space action and the Humanoid environment.

The expected results for the project are the trained models with a learning that shows a learned behavior, the code of training-testing of the environments and a written report describing the algorithm and the design process.

Deadline: 03/02/2020 11:59 PM CET

2. Proximal Policy Optimization

The family of algorithms called Proximal Policy Optimization, is a group of reinforcement learning methods belonging to the Policy Gradient optimization field, which seek to provide a new alternative of learning taking the benefits of the Trust Region Policy Optimization TRPO method but simplifying its implementation.

The main problem of the general Policy Gradient is that they work calculating a gradient through automatic differentiation starting from the loss function, approach that suffers the implications of possible large policy updates

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$

Equation 1. Standard loss function of the Policy Gradient methods.

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$

Equation 2. Gradient estimation through Loss differentiation.

With the purpose of propose a new set of approach, the TRPO[2] defined a "surrogate" objective and constrain the maximize calculation of the parameters of the new policy given an advantage estimation at a defined time stamp.

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta \end{aligned}$$

Equation 3. Maximization of the surrogate constrained.

I. CLIPPED SURROGATE OBJECTIVE

Having defined $r_t(\theta)$ as the surrogate objective, and aiming to move this value from the old policy parameters represented by 1 in this notation, $r_t(\theta) > 1$ means that the action is more probable in the current policy, meanwhile $0 < r_t(\theta) < 1$ denotes that the action is less probable in the current policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

Equation 4. Defined surrogate objective as the ratio of policies.

The Clipped Surrogate Objective clip the probability ratio given a parameter epsilon and compare it with the unclipped objective, taking the minimum between them. This clipping penalizes the moving of $r_t(\theta)$ outside of the interval created by the parameter epsilon.

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

Equation 5. Maximization of the surrogate constrained.

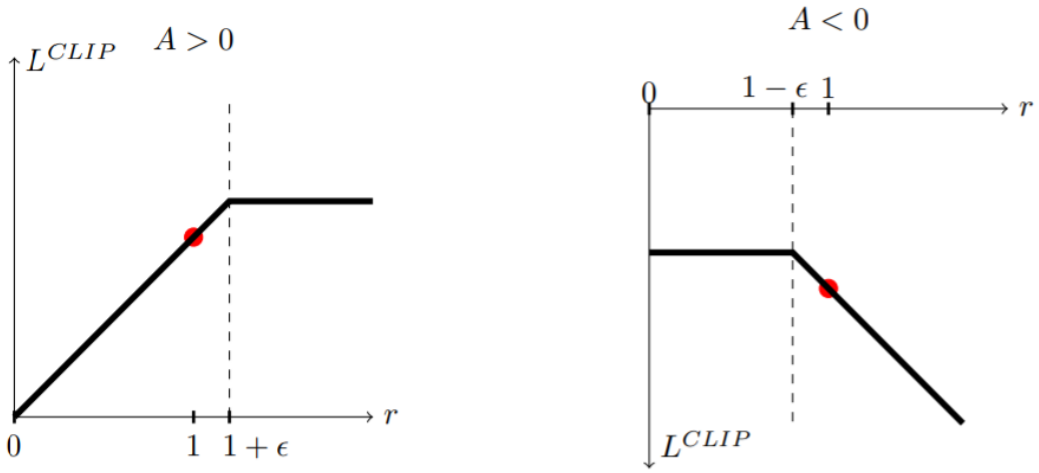

Figure 1. Surrogate function L^{CLIP} at a specific time step with respect to the value of $r_t(\theta)$ and the sign of the advantages + (left) and - (right)[1].

This dynamic represented by the equation 5 is interpreted based on the advantage estimation, according to which if the advantage is positive the probability of take that action tends to be increased and if it is negative it tends to be decreased. Those changes over the policy are continuously controlled by the clipping factor, which discourages large updates out of the comfort zone given by epsilon.

Along with the training of the models for the action is also given the entropy factor with all the dynamics associated with it that allows the exploration process to reach the most possible configurations.

II. ALGORITHM

In order to implement the Proximal Policy Optimization is needed:

1. Define an Actor-Critic model composed by neural networks.
2. Obtain a calculation of the returns using the Generalized Advantage Estimation which reduces the variance with a smoothing factor $0 < \lambda < 1$ with a suggested value of 0.95.
3. Calculate the surrogate policy loss.
4. Mini-batch updates using a fixed number of epochs.

Using this function is possible to update the parameters in the Neural Networks, following the algorithm described in the paper.

Algorithm 1 PPO, Actor-Critic Style

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

Algorithm 1. General approach for the Actor-Critic model learning based in PPO[1].

3. Implementation

Since the main challenge of implementing the PPO algorithm over the given requirements, is the coding it in a new release of TensorFlow which reached its second version in the final months of the year 2019, it was needed a reference implemented in its previous version.

The base structure of the loss calculation was taken from a Proximal Policy Optimization Algorithms – PPO blog[4] and from the baselines of the OpenAI repository[3]. From this start point it was added interfaces to control the management of the training loop, the files and the results logging, to finally implement the solution over the environments Mountain Car in continuous action space and Humanoid.

The highlights of the development of the code are focused mostly on the use of Gradient Tape for the Loss calculation, the backup of the information through the use of Tensor Board and some useful features of the Objected Oriented Programming, facts that can be inspect in the code an repository of the project.

4. Results obtained

Having implemented the code and standardized the flow of the information along the project, the parameters related to the algorithm of the PPO model update were fixed with the following values:

POLICY CLIPPING FACTOR = 0.2
VALUE CLIPPING FACTOR = 0.2
ENTROPY COEFICIENT = 0.0
MINI-BATCH SIZE = 25
PPO EPOCHS = 10
STANDARD DEVIATION = 1.0
EPSILON = 0.9

From there the main changes done to the training process were the number of epochs during of which information would be sampled, and the size of the training batch used for the learning.

I. MOUNTAIN CAR CONTINUOUS SPACE

For the mountain car environment in the continuous action space, the actor model is defined by a three layers neural network with 128, 64 and 1 output unit with tanh activation function, while for the critic model 1 output with linear activation function.

The figure 1 shows the reward sampled from the exploration phase which denotes a low level of dispersion in the final reward of each episode, subsequently the figure 3 shows a convergence reaching the value 75 as an average reward.

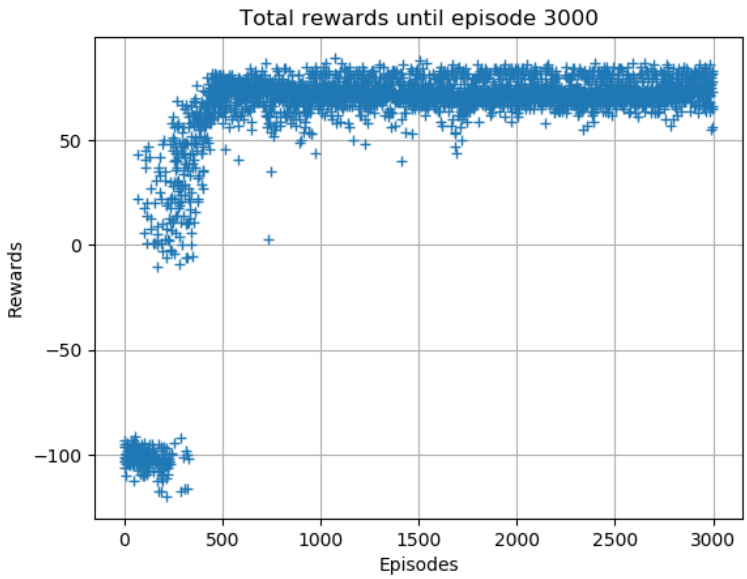

Figure 1. Reward sampling over the exploration phase in the Mountain Car environment.

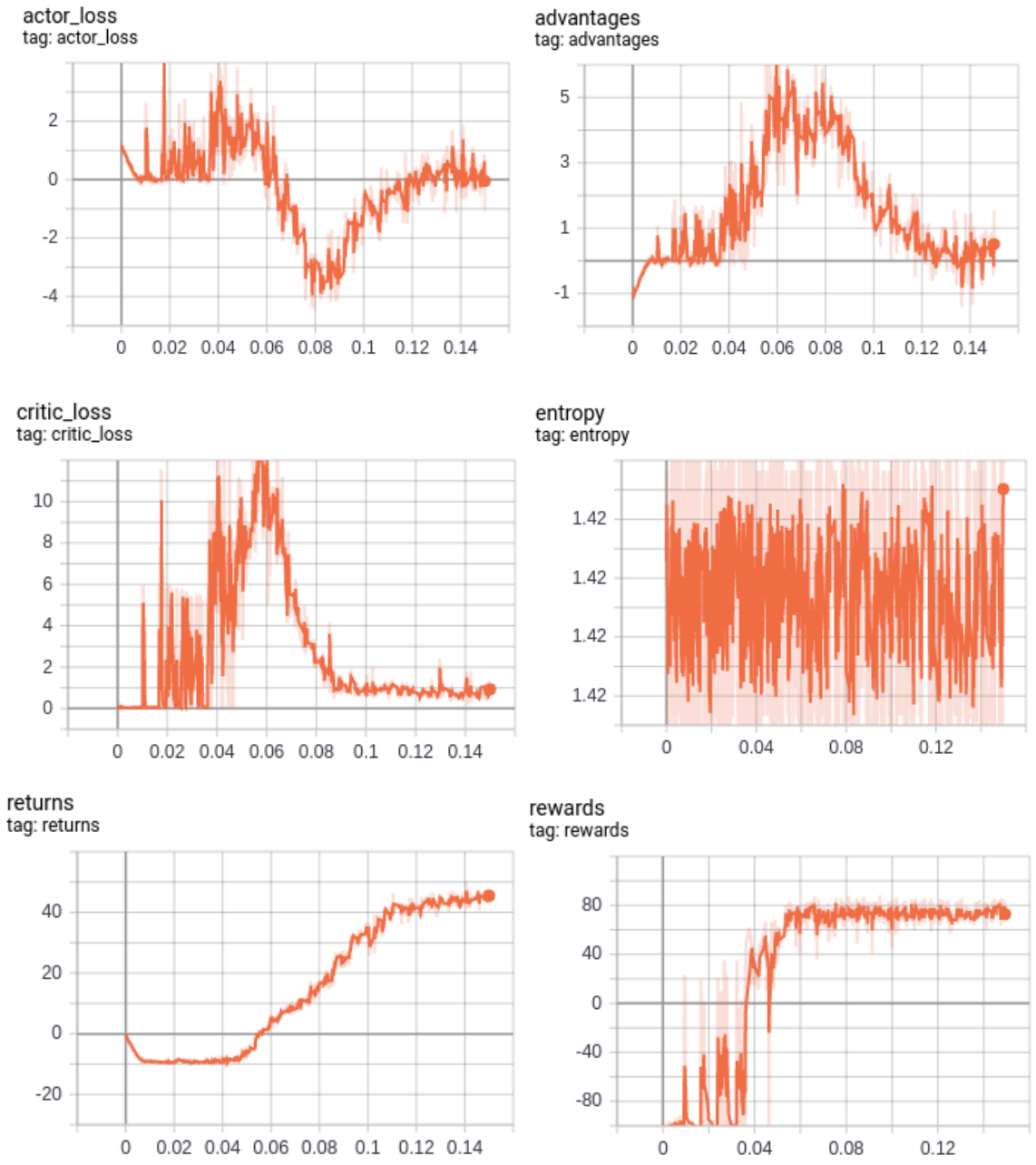

Figure 2. Set of curves for monitoring the training of the Mountain Car environment extracted from Tensor Board.

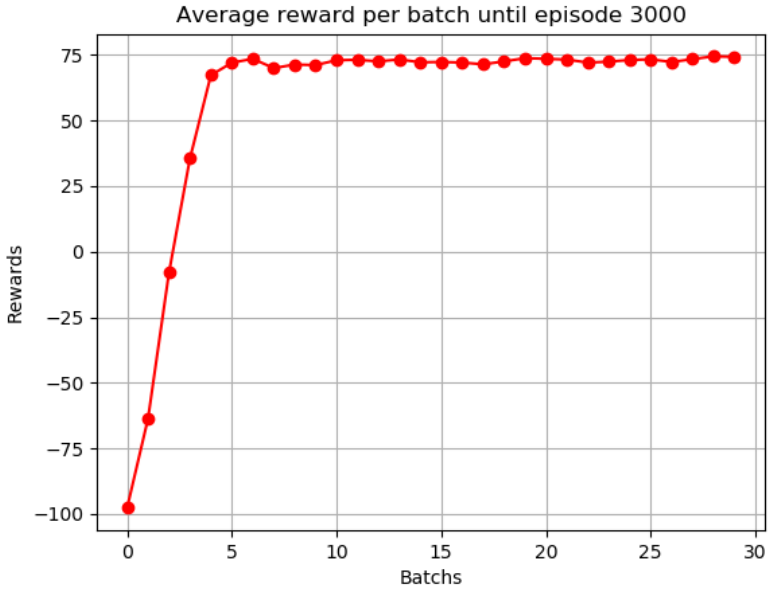

Figure 3. Final reward average over the exploration phase in the Mountain Car environment.

II. HUMANOID

For the mountain car environment in the continuous action space, the actor model is defined by a six layers neural network with 512, 256, 256, 128, 64 and 1 output unit with tanh activation function, while for the critic model 1 output with linear activation function.

In contrast to the Mountain Car environment the figure2 depicts a much greater dispersion of the rewards sampled, reaching a peak around episode 3000 and losing it over further training. Based on that information, the model with the greatest reward average was taken for the test phase.

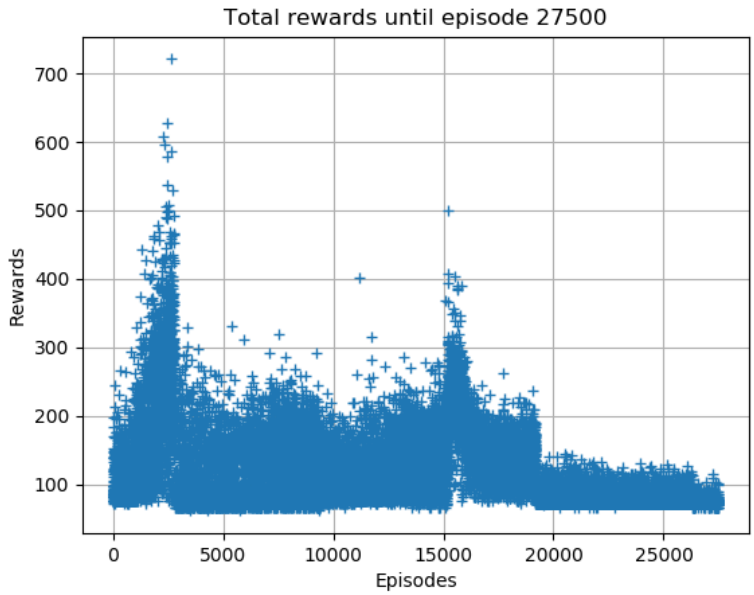

Figure 4. Reward sampling over the exploration phase in the Humanoid environment.

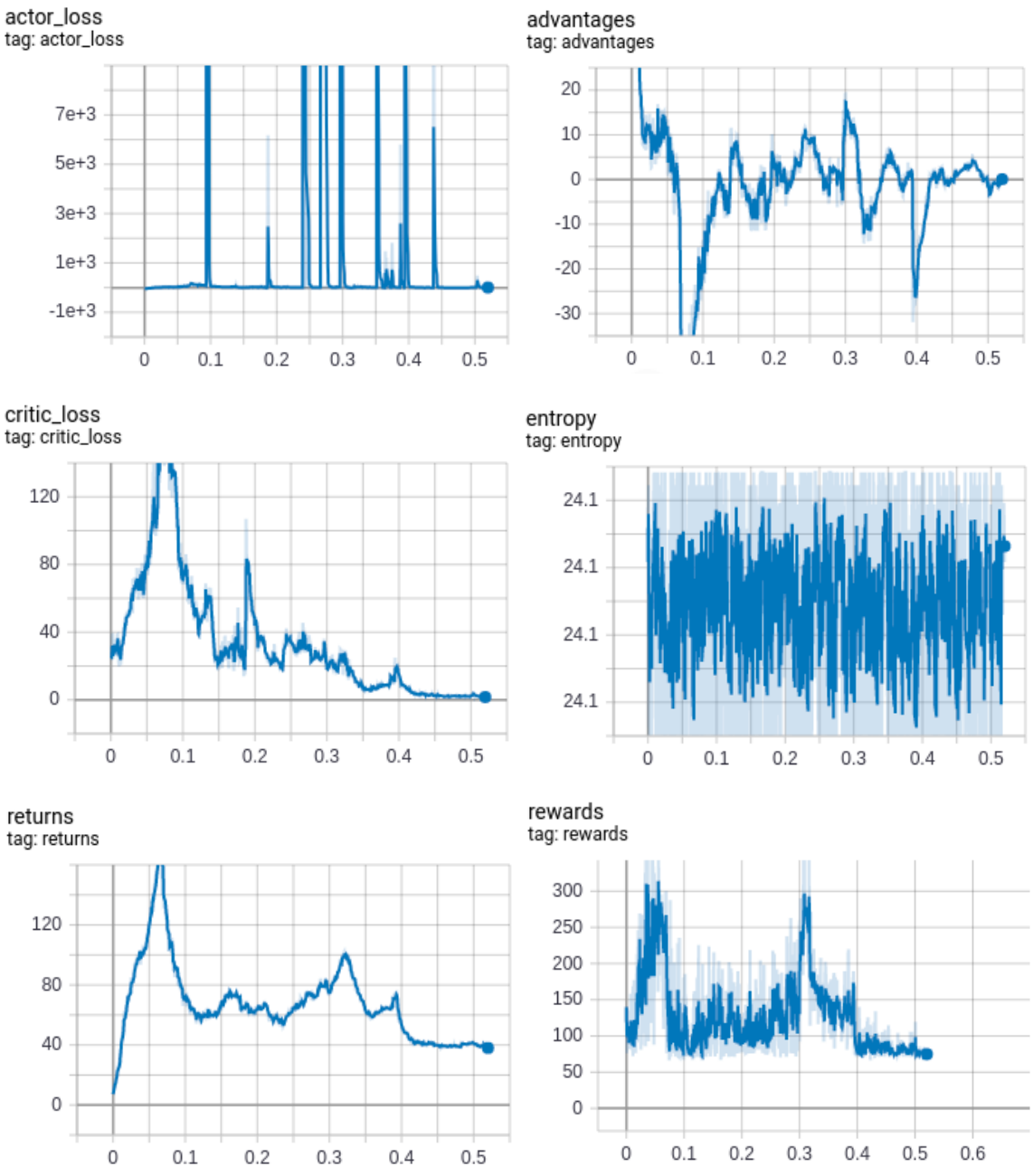

Figure 5. Set of curves for monitoring the training of the Mountain Car environment extracted from Tensor Board.

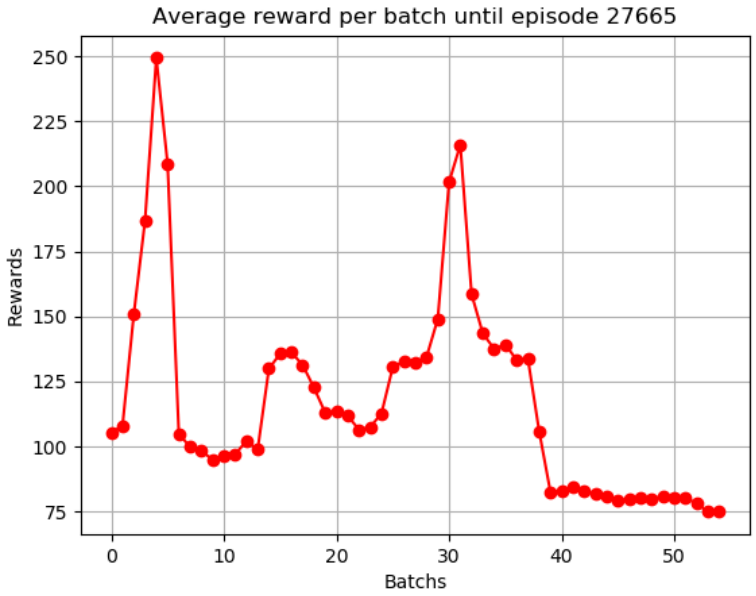

Figure 6. Final reward average over the exploration phase in the Humanoid environment.

5. Problems during implementation

During the implementation the first approach was trying to optimize the gradient calculation function directly embedded inside in the Keras model, passing the method to the Adam optimizer. This first implementation holds the mathematical background, but due to the new features of the framework Tensor Flow 2 the default eager execution needs to be disabled in other to be able to handle the Tensors defined for the Loss calculation.

Nevertheless, disabling the eager mode would compromise the performance of CPUs and GPUs drastically slowing the execution of the learning process. Therefore, is needed to define a Gradient Tape and explicit calculate the gradient and apply it to the model using a relatively low-level programing.

6. Conclusions

The implemented version of the Proximal Policy Optimization algorithm using the Clipped Surrogate Objective depicts positive results on loss reduction given an Actor-Critic model.

In the case of the Mountain Car in the continuous space action, the loss reduction of the actor and the critic trend to cero the further the training went. Meanwhile, the returns and the rewards share the same behavior, last of which converge to a fixed value reaching a great policy to control the car movement towards the goal.

For the Humanoid environment a different behavior is displayed by the loss curve in the actor model which peak along the training, on contrast the loss of the critic model which tends to 0, along with an advantage function that keeps oscillating. The result of the training consists in improving the policy to reach a rewardable behavior, which in the present case is advancing a few steps and aim to push the humanoid body as far as possible in the simulation.

For both environments the entropy kept oscillating over all the training.

The models, the log files and the videos taken of the environments are gathered in the repository of the project[6].

7. References

[1] SCHULMAN, John, et al. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[2] SCHULMAN, John, et al. Trust region policy optimization. En International conference on machine learning. 2015. p. 1889-1897.

[3] SCHULMAN, John, et al. Proximal Policy Optimization Blog 2017 https://openai.com/blog/openai-baselines-ppo/

[4] JAYASIRI, Varuna. Proximal Policy Optimization Algorithms - PPO Blog http://blog.varunajayasiri.com/ml/ppo.html

[5] HUI, Jonathan. RL — Proximal Policy Optimization (PPO) Explained Blog 2018 https://medium.com/@jonathan_hui/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

[6] https://github.com/fhfonsecaa/RL_PPO