Continuous Control Project

# A continuous Deep Reinforment Learning case study : Unity Reacher environment

Gabriel Colas

2022

**Abstract**

The pairing of deep neural network and Reinforcment Learning and its improvements has outperformed the historical methods on a large panel of environment. In this work we applied a state-of the art policy-based method to directly learn the optimal policy. We used the proximal policy optimization technique actor-critic style to the benchmark Unity game environment Reacher. We have demonstrated great performances and a great stability with this method. Indeed, the game environment is considered solved with a fast learning curve.

## 1 Introduction

Reinforcement learning is a framework that allows a smart agent to act with a provided environment and find the optimal policy within the environment. Since we don't know the intrinsic dynamic of the Markov Decision Process we discover the policy by sampling its dynamic. Thus, the historical idea derived from dynamic programming is based on the famous recursive Bellman Equation that led to a variety of algorithms commonly used for a learning agent [O'Donoghue et al., 2018].
However, there is also a more direct way to learn an optimal policy. Rather than focusing on learning a state values or states action values could we straight learned the policy ? This is at the heart branch field in RL named Policy-based methods where we learn directly an optimization-based optimal policy. Many algorithms have been suggested by combinatorial optimization as Hill-climbing, Cross entropy method or evolutionary strategies. But gradient-descent based algorithm have made the success of this field as a faster way to learn a good policy, more robust and easy to use with back-propagation or graph based optimization methods widely used for Neural Network. The REINFORCE algorithm from 1992 [Williams, 1992] has been a profound achievement before more data-efficient method has been created.
On an other hand, dealing with large continuous states and actions has been a tough barrier that has been tackle first by [Mnih et al., 2013], with the power of Neural Network generalization. Combining a deep neural network and the usual Q-learning algorithm has considerably enhanced the benchmark scores on the suite of controlling ATARI games and outperformed significantly human-based performance. Since the kick-start of this new Deep-RL field many improvement and variant have been suggested and outperformed the DQN algorithm.
In this case study, we will implement the core of the Proximal Policy Optimization algorithm [Schulman et al., 2017] more data-efficient than the REINFORCE algorithm with minor changes to deal with the continuous action spaces of the Unity Reacher environment

## 2 Theory

During this work, we made several attempts to find the optimal policy for the Reacher environment. We describe here the PPO actor-critic algorithm we used and its underlying theory.
We consider a smart agent that take an action $A_t$ in an environment at each time step $t$. The agent interact with this environment that return an observation $O_t$, which we consider for simplification to be its internal states $S_t$. The environment is formally described by a Markov Decision Process where the current state $S_t$ completely characterize the process. The goal is to find the optimal policy $\pi^*(s|a)$ by maximizing the expected total reward. We consider in this work, that we sample a stochastic policy from a normal law $a_t\ \pi(s_t|a_t)$ with $\pi(s_t|a_t) \sim \mathcal{N}(\mu,\ \sigma^2)$. We describe the problem as an gradient optimization algorithm and a policy-based method as an episodic-return objective of the form:

$$g = E[\sum_{t=0}^{\inf} At \nabla_\theta \pi_\theta(s_t|a_t)]$$

In this objective we have willingly omit the discounted $\gamma^t$ of the expected return by introducing a little bias. At this point this is the core algorithm of REINFORCE. The gradient ascent algorithm over the parameter $\theta$ of function approximation fit the corresponding weight to find the optimal policy at each time step. It's the "Vanilla" policy gradient method that could be extent to other formulation of the gradient as we will explain here of the surrogate function. We derived our objective from the trust region policy optimization algorithm (TRPO) that still challenging when implementing the gradient ascent over the parameters. Also even if monotonic increase to the optimal policy is proved to be a propriety of this algorithm it can still be really long to find out. So we derived the surrogate objective from importance sampling. It is more data-efficient than collecting trajectories and optimize once along each one.

$$L(\theta) = \hat{E}[\hat{A}t \frac{\pi_{theta}(s_t|a_t)}{\pi_{\theta old}(s_t|a_t)}]$$

However, a constraint is necessary to avoid a too large policy update :

$$L(\theta) = \hat{E}[min(\hat{A}t \frac{\pi_{theta}(s_t|a_t)}{\pi_{\theta old}(s_t|a_t)}, clip(\frac{\pi_{theta}(s_t|a_t)}{\pi_{\theta old}(s_t|a_t)}, 1-\epsilon, 1+\epsilon)\hat{A}t)]$$

Then for this policy-gradient method we need to find an estimate of $\hat{A}t$. The easier estimate would be the discounted sum of reward. However, closely equivalent to the Monte-Carlo estimate we now that it is inclined to a large variance. It could destabilize the learning and even reduce the capability of convergence. Thus, we introduce the Proximal Policy Optimization as an actor-critic method by reducing the variance by a baseline. It does not change the convergence of the surrogate because of the intrinsic linear property of the expected value.

$$\hat{A}t = \sum_{t=0}^{\inf} \gamma^t r_t Future - b(s_t)$$

We could then estimate $\hat{A}t$ by a lot of different way to reduce variance. We decided first to use a value estimate and the TD-Error but then we improved the algorithm by using the generalized advantage estimation [Schulman et al., 2015] :

$$\hat{A}t = \sum_{l=0}^{\inf} (\gamma\lambda)^t \delta_{t+l}$$

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

A variety of different methods can then be used to estimate the value function. When we use a nonlinear function approximation we can learn the parameters by minimizing the mean squared error between the predicted value estimate and the estimated value estimate as the discounted sum of reward.

$$\min_{\Phi} \sum_{n=1}^{N} ||V_{\Phi}(s_n) - \hat{V}_n||^2$$

Finally, for now this method in only adapted for discrete action space. Some minor changes are needed to adapt the algorithm for a continuous task. We applied a commonly and trivial technique by adding a stochastic sampling from a Gaussian Policy. So our actions are sampled from this. We get then :

$$\pi_\theta(a_t, s_t) = \mathcal{N}(\mu_\theta, \sigma^2)$$

# 3 Methods

For these experiments the program run in a laptop with a GPU GTX 1050. We learned our neural networks with Torch and the GPU for python 3.6. We give open access to the python code in the Git-Hub [student, 2022].

The environment provides a continuous vector observation of space size 33 for 20 parallel robotic arms. Each arms is controlled by 4 continuous actions in the range [-1,1] that represent torque applied to the two joint arms. We built two neural network as nonlinear function approximation respectively the actor network of $\mu_{theta}$ and the critic network to learn a value estimate.

The algorithm is shared on two steps. First we collect the parallel trajectories by using our approximate optimal policy by sampling the Gaussian law from the actor network, we calculate the old probability from this behavior and the advantage estimate from the evaluation of the critic network. Then we applied the training process described during the first section and shown here :

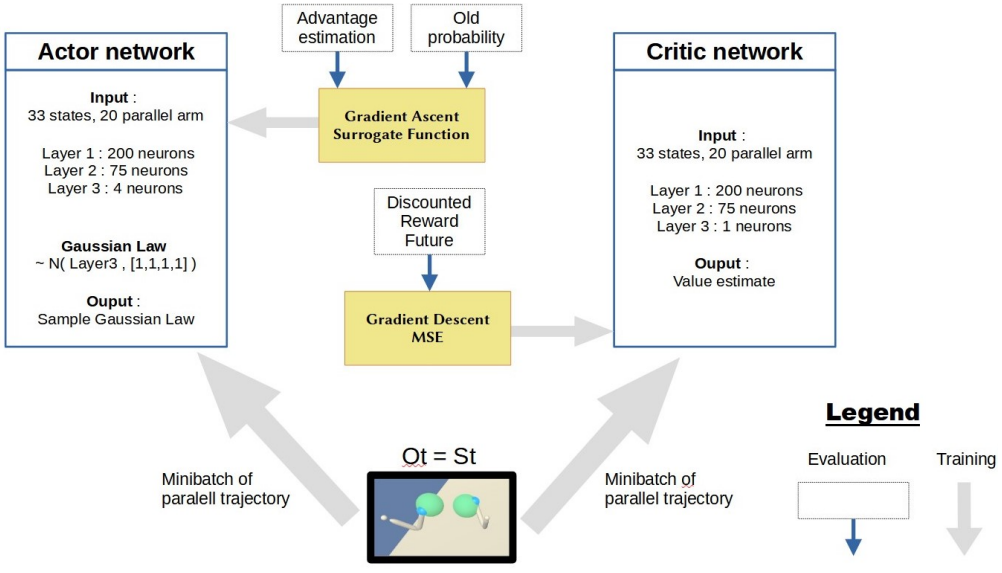


Figure 1: Training process for the PPO Actor Critic algorithm

During the several attempts of this work the experiment, the optimal use of the data collected during the evaluation part of $t = 1000$ is to shared the data in sampled of minibatch of size 64 without shuffling the data inside. Then we apply 8 epoch over each minibatch of the collected trajectories

We have kept the different hyper-parameters for the PPO algorithm as following :

| Model | $\gamma$ | $\lambda$ | Batch size | Epoch | Actor lr | Critic lr | Tmax | $\beta$ | $\epsilon$ |
|---|---|---|---|---|---|---|---|---|---|
| PPOActor-Critic | 0.9997 | 0.95 | 64 | 8 | $2e-4$ | $2e-4$ | 1000 | 0.01 | 0.1 |

## 4 Results and Analysis

We present the performances of our optimal value-based algorithms according to the parameters drawn from the table last section. We show two different attempts over the same experiment setup. We see that for both experiment, the environment is considered solved (It needed a mean reward $> 30$ over 100 episodes), and the final episodes performance are over 30.
However between the two experiment there is over 80 episodes differences before the the environment is considered solved. A different instantiation of the random seed can produced drastic changes in the performances. Also we found that in the lower performance experiment, the critic network didn't learned any value estimate as we see in 3 from Tensorboard. So this experiment were completely critic-free it was a policy-gradient straight method. And of course we knew that policy-gradient method without the baseline would be less data-efficient.

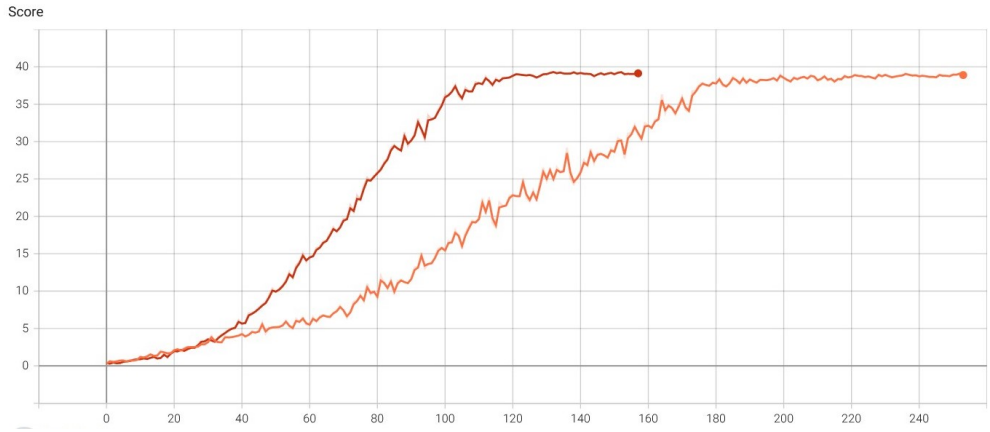


Figure 2: Performances of the two experiment in the reacher environment, orange light curve: Experiment with critic network missed learning

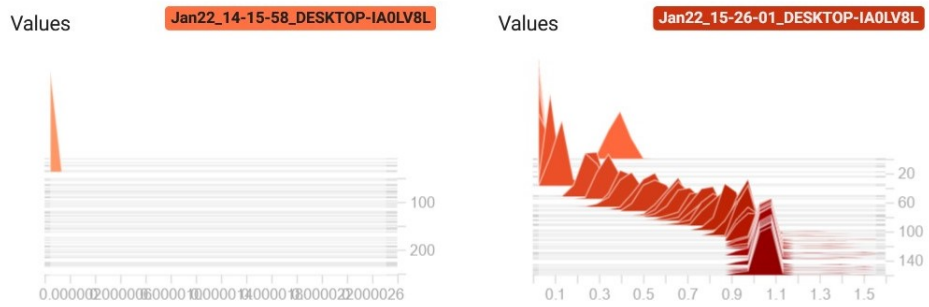


Figure 3: Value estimate distribution of the two experiment in the reacher environment during the training, left histograms: Experiment with critic network missed learning

## 5 Conclusion

In this work we learned to 20 parallel robotics arms to follow the green ball and achieve maximum reward in the more data-efficient way. However some improvement are still possible to improve the stability and the use of our collected trajectories. We could follow the work from [Hämäläinen et al., 2020] guided by entropy method combined with evolutionary strategies to find the optimal variance of our Gaussian Law here fixed at 0.5.

## References

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Gabriel PHD student. Project navigation. https://github.com/GabrielLinear/PPO, 2022.

Perttu Hämäläinen, Amin Babadi, Xiaoxiao Ma, and Jaakko Lehtinen. Ppo-cma: Proximal policy optimization with covariance matrix adaptation. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2020.