ULIEGE UNIVERSITY

Optimal Decision Making For Complex Problems

INFO8003

Final Project. Inverted Double Pendulum: Searching

High-Quality Policies to Control an Unstable Physical System.

Authors:
Aleksei Korneev s208004
Ruslan Alagov s208098
exchange students
group 13

Professor :
D.Ernst
Assistants :
S.Aittahar
B.Miftari

14 May 2021

1 Introduction

In this project, we were asked to build a smart agent that keeps in equilibrium a Double Inverted Pendulum, illustrated in Figure 1. The agent interacts with the environment by applying an horizontal force on a cart in which a two-link pendulum is mounted on its center and is initially set to (nearly) upright position above the cart. At each transition, additionally to the reward given by the environment, the agent receives a positive, constant signal. A terminal state of the environment is reached when the distance between the upright and the current state is above a given thresold.

2 Tasks

1) As first task, provide a formalization of the Double Inverted Pendulum environment (excepted the dynamics) based on the given source code. Please note that what the agent observes is not necessarily a state of the environment. Provide also characterizations of this environment (e.g., deterministic or stochastic...). The format of your formalization should be inspired from the paper “Tree-based batch mode reinforcement learning”. You should also install all needed dependencies to be able to exploit the environment source code.

2) Design your own policy search technique based on the reinforcement learning literature. Provide references upon which paper your approach is based on. Your algorithm has to be able to learn both discrete and continuous policies. Design an experimental protocol which assesses the performance of your algorithm with both policies, com-pared to a classical algorithm seen during the lectures (e.g., FQI with ensemble of trees). Display the performance of your policies at the end of each episode in terms of expected discounted cumulative reward.

3 Domain

Double Inverted Pendulum Environment is a deterministic environment. State space contains of 6 continuous variables:

slider: position[-1:1], slider velocity;

pole1: theta[-pi:pi], theta velocity;

pole2: gamma[-pi:pi], gamma velocity.

Action space contains 1 continuous variable: force[-1:1].

Reward is the sum of alive bonus and distance penalty. The alive bonus is equal to 10 and distance penalty is calculating according to the following formula:

$dist\_penalty = 0.01 * (pos\_x)^2 + (pos\_y + 0.3 - 2)^2$,

where pos_x and pos_y are coordinates of the upper pole position. One obtains the maximum reward when the position of pendulum is close to the upright one.

4 Policy search technique

Our policy search technique is based on the deep deterministic policy gradient (DDPG) algorithm. This approach uses actor-critic approach based on the deterministic policy gradient (DPG) algorithm. It contains 4 neural networks: deterministic policy network, Q network and two target networks that slowly track these learned networks. Also DDPG contains the replay buffer that allows obtaining the i.i.d samples for training. Exploration challenge of learning in continuous action spaces is solved by adding noise sampled from a noise process, in our implementation we used an Ornstein-Uhlenbeck process to generate temporally correlated exploration. Detailed description of the algorithm is provided in [1].

To obtain the discrete policy by using of DDDPG we use an algorithm that is based on Wolpertinger Architecture[2]. Due to the fact that Inverted double pendulum action space is one dimensional we decided to implement our own technique to find nearest actions instead of using K-NN algorithm. In our implementation we simply have discretized the action space to the following set of actions [-1, -0.5, 0, 0.5, 1]

and choose two nearest actions to the action obtained from actor network by brute force.

We have used 4000 epochs and set the maximum transition number equal to 500. The size of replay buffer is equal to 1000000. We also set following values for parameters of DDPG: size of minibatch = 128, gamma = 0.99, tau = 0.001, actor neural network learning rate = 0.0001, critic neural network learning rate = 0.001. Our implementation of actor network contains three layers, where the number of neurons for the first layer is equal to the number of network input dimension (number of observational variables), the number of neurons for the second and third layers are equal to 256. Critic network has more complex structure, the first layer is used to accept the state vector, the second layer obtains the output of the first layer and the vector of action, the last two layers simply follow each other. The number of neurons in hidden layers are equal to 256.

Further we consider the results of execution of our program, all charts below represent expected discounted cumulative reward. Blue line represents the expected policy, the orange one – average for the last 10 epochs. Reward and discounted reward are obtained in different executions. That's why increasing of reward is noticeable for different number of episodes. The results of the execution of algorithm for continuous policy are shown at the Figure 1 and Figure 2. As we can see, after 2000 of episodes (epochs) our policy allows us to increase the time of keeping the inverted pendulum in upright position and thus to get larger reward. According to our observations training process is not stable and larger rewards usually occur from 1000 to 3000 episodes and gradually increase.

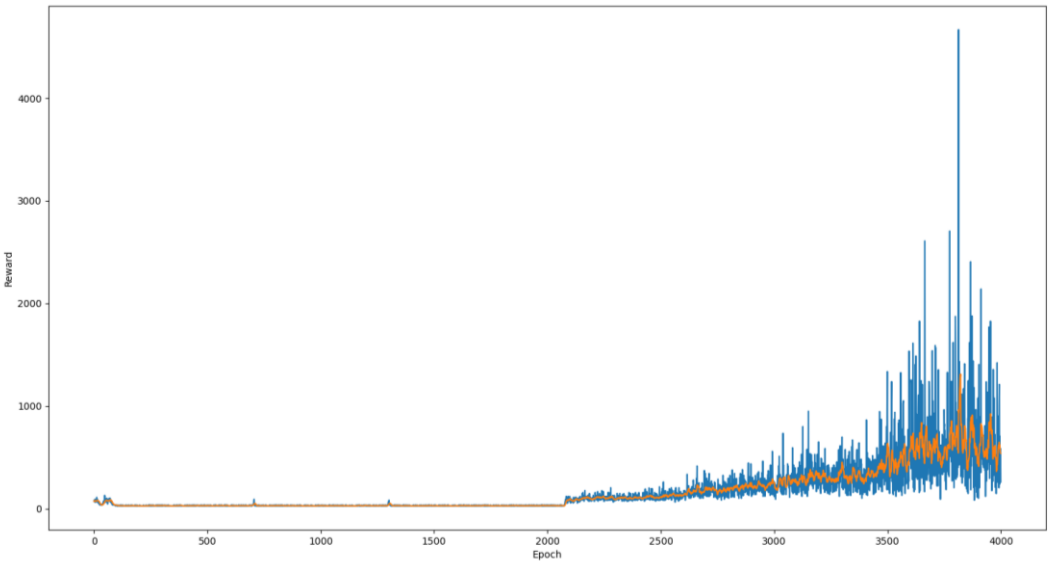

Figure 1. Continuous policy reward.

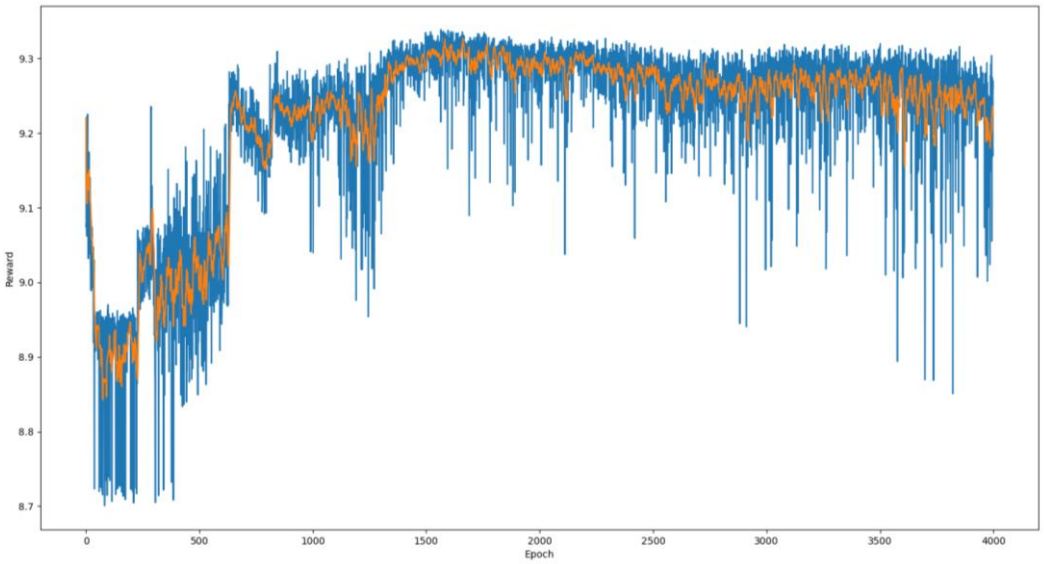

Figure 2. Continuous policy. Discounted reward.

The result of execution of program for discrete policy is shown at the Figure 3 and Figure 4. Due to the fact that training of neural network is unstable and contains noise we can observe a pick of the reward, but then it comes back to normal values, reward gradually increases, but worse than for continuous policy. We can explain this result by the fact that continuous policy is able to control the pendulum more precisely to compare with "rude" force of discrete policy.

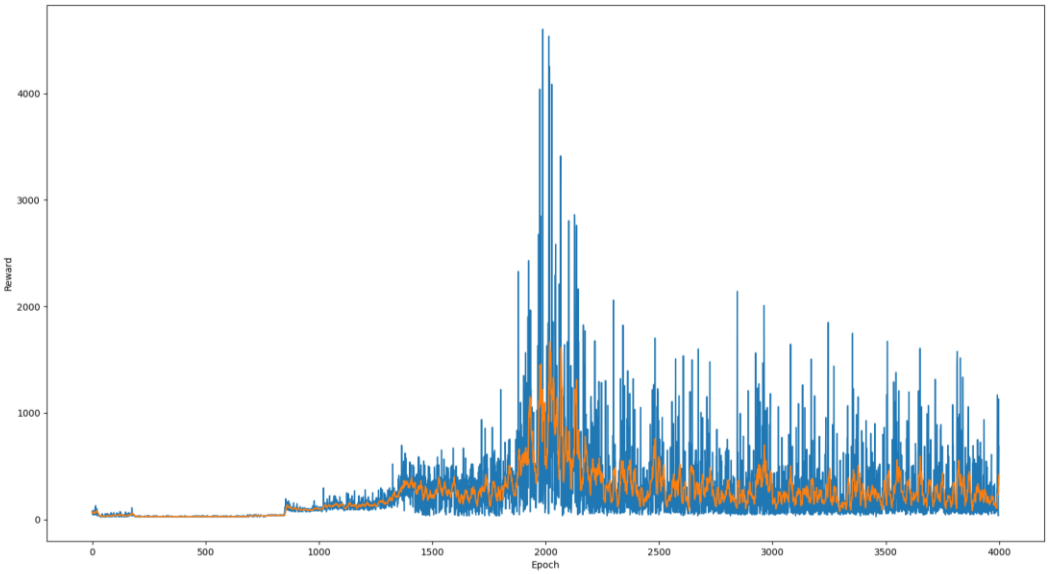

Figure 3. Discrete Policy reward.

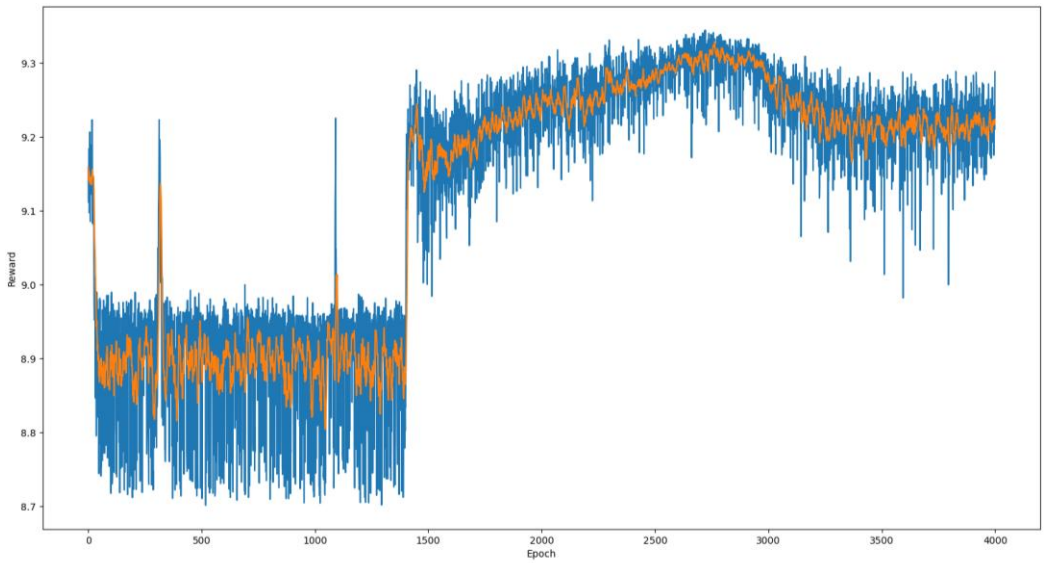

Figure 4. Discrete Policy. Discounted reward.

The Figure 5 shows the result of fitted Q-iteration algorithm for extremely randomized trees regression method. We use random policy for the first 1000 of episodes, then we trained our model with depth = 100, and applied it for the next episodes. We can notice that after the moment we started to use trained model the average reward is larger, it means that our model is able to control pendulum more time than random policy, but this result is more poor than for the discrete case of DDPG.

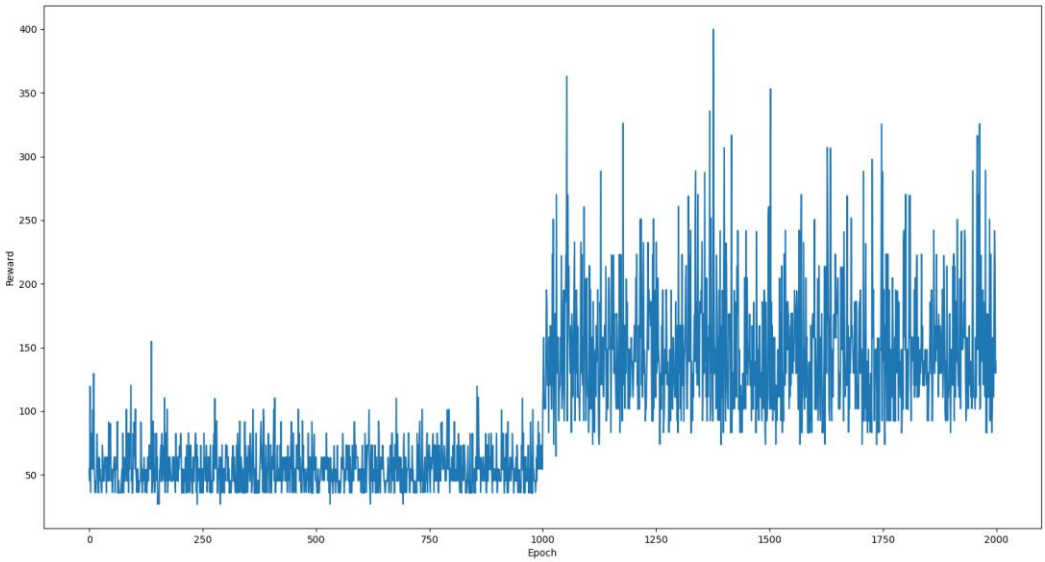

Figure 5. Extremely randomized trees reward.

Our approach is based on the DDPG, but it can be improved, for example, via Prioritized Experience Replay technique, which is described in [3]. Another way is an algorithm that extends DDPG to bootstrapped actor-critic architecture [4].

References

1. Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

2. Gabriel Dulac-Arnold*, Richard Evans*, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt,Timothy Mann, Theophane Weber, Thomas Degris, Ben Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. arXiv:1512.07679v2 [cs.AI] 4 Apr 2016.

3. Yuenan Hou, Yi Zhang. Improving DDPG via Prioritized Experience Replay. DOI:10.13140/RG.2.2.23694.41287.

4. Zhuobin Zheng, Chun Yuan, Zhihui Lin, Yangyang Cheng, Hanghao Wu. Self-Adaptive Double Bootstrapped DDPG. https://doi.org/10.24963/ijcai.2018/444.