DDPG algorithm applied to testing environment 'MountainCarContinuous' and to experimental environment 'Swimmer robot' in continuous action space

Vannoli Marco matr.1860363*
Master's degree in artificial intelligence and robotics Department of computer engineering, automatic e management "Antonio Ruberti" University of Sapienza Roma
(Dated: February 2, 2020)

1. INTRODUCTION

The work aims to address a problem through a method or algorithm based on Reinforcement Learning. This problem, as will be described in this paper, concerns a very specific task that takes place in a continuous space and involves reaching a machine on the mountain peak where a flag is placed (it will be exposed in the "Task" section 5, "Mountain car in continuous space"). The algorithm which is taken on consideration is the **Deep Deterministic Policy Gradient** which, for these types of tasks that take place in a continuous space, obtains, as we will see, good learning results. Furthermore, the exposed work, purposes to applicate the algorithm on both a testing environment and experimental environment, in particular, this last, based on an agent's learning of swimming skills in a continuous environment. In this paper, initially, a notion of Deep Reinforcement Learning will be given to then introduce the **DDPG** algorithm both from a theoretical point of view (section 3) and from an application point of view (section 4). In this last point, in precise, the structure with which the algorithm in question has been implemented will be analyzed. In the penultimate sections 6-7, the results obtained in both the test environment and in the experimentation environment will be shown to analyze the algorithm at 360. Finally, conclusions and future considerations will be given regarding the work done (last section 8).

2. BACKGROUND: DEEP REINFORCEMENT LEARNING

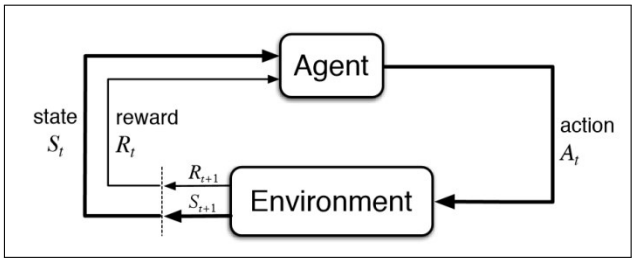
an exhaustive definition of Deep Reinforcement Learning is the following: *Deep reinforcement learning links artificial neural networks with a reinforcement learning structure that allows software-defined agents to learn the most beneficial actions possible in a virtual environment to achieve their goals.*Deep reinforcement learning joins artificial neural networks with a reinforcement learning structure that allows software-defined agents to learn the most beneficial actions possible in a virtual environment to achieve their goals. Given this definition, it is automatic to introduce Reinforcement Learning which represents a branch of learning and can be represented as a black box in which only inputs and outputs are shown. (see figure 1). In fact, Reinforcement Learning often is applied in fields in which the environment's functions are unknown and its aim is to learn a series of actions that will guide an agent to achieve its goal. It is important to describe the factor which corresponds to Reinforcement Learning Problem:

- An **agent** takes actions An **action** which represents the set of all possible moves the agent can be taken.
- An **Environment** or world trough which the agent moves and which respond to the agent.
- A **discount factor** designed to make future rewards worth less than immediate rewards.
- A **state** that presents the situation in which the agent is.
- A **reward** is the feedback with which the system measures the success or failure of actions taken by the agent in a given state.
- A **policy** is the rule that the agent applies to determine the next action based on the current state.

Reinforcement learning stages an agent's attempt to approximate the environment's function, such that we can transfer actions within the black-box environment that maximizes the rewards it picks up.[1]

* vannolimarco@hotmail.it

FIG. 1. **the "black box" of Renforcement Learning**



3. DEEP DETERMINISTIC POLICY GRADIENT

Deep Deterministic Policy Gradient, also known as the abbreviation **DDPG**, is an algorithm that purpose to learns a Q-function and a policy. It is important to note that the goal of Q-learning is to learn a policy that shows the agent what action to take under what circumstances. Thus, the Markov decision process, Q-learning attains a policy that is optimal that it maximizes the expected value of the total reward over any successive steps, starting from the current state. Coming in back to DDPG, the algorithm uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. The Learning of Q-learning involves knowing the important connection between action-value function $Q^*(s,a)$ and the optimal action $a^*(s)$: if we have Q^*, we can directly obtain the optimal action, $a^*(s)$, via

$$a^*(s) = \arg\max_a Q^*(s,a).$$

The key concept of DDPG is to perform a learning an approximator to $Q^*(s,a)$ through learning an approximator to $a^*(s)$ for environments with continuous action spaces. However, the fact of working in a continuous space of actions turns out to be more difficult than in a discrete space of actions since from a point of view of calculating the maximum on actions in $\max_a Q^*(s,a)$. Therefore, due to the continuity of the space of the actions the evaluation is compromised and there is the need to approximate this calculation using a learning based on the gradient for the policy:

$$\max_a Q^*(s,a) \approx Q(s,(s))$$

In order to better understand the algorithm, it is necessary to go in-depth on two important parts that make up the DDPG algorithm: the learning Q-learning (see subsection Q-learning) and the Policy learning(see the Policy Learning).

The Q-Learning

The Q-learning is an off-policy reinforcement learning algorithm that attempts to find the best action to perform given the current state. It's regarded off-policy because the q-learning function learns from actions that are outside the current policy. In a few words, it aims to seek to learn a policy that maximizes the total reward. In our case, in the use of DDPG, the q-learning algorithm is used as a function approximator and its purpose to minimize the mean-squared Bellman error (**MSBE**) loss function. The starting point is The Bellman equation that has the fundamental role of learning an approximation to the optimal action-value function $Q^*(s,a)$ described by it the following expression:

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s'}[r(s,a) + \gamma \max_{a'} Q^*(s',a')]$$

(*$s' \sim P$ = *the next state where s' is sampled by the environment from* $P(\cdot|s,a)$).
If the approximator is a neural network $Q_\phi(s,a)$, with parameters ϕ and a set D of transitions are collected (s,a,r,s',d^*), then is possible introduce the mean-squared Bellman error (MSBE) function with which is potentially possible evaluate how closely Q_ϕ arrives to satisfying the Bellman equation:

$$L(\phi,\mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

(* notes if the state s' is terminal)

The Q-learning, not only for the algorithm as DDPG, employees of a specific technics with which approximate $Q^*(s,a)$:

- **Target Networks**: the term

  $$r + \gamma(1-d)\max_{a'} Q_\phi(s',a')$$

  defines the **target** factor with which it is possible to minimize the **MSBE** loss attempting to make the Q-function be more like this entity. The issue which arises is that the target is correlated by the same parameters that need to train ϕ. Thus is necessary to define a set of parameters that are close to ϕ with a time delay and this fact brings to build a second network (target network) denoted by ϕ_{targ} parameters. In DDPG scenario the target network is updated through the main network update by Polyak averaging:

  $$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi,$$

  ρ is a hyperparameter between 0 and 1 called polyak or tau (τ).

- **Replay Buffers**: The replay buffer is used to get a stable behavior of the algorithm about approximator $Q^*(s,a)$ and consists of the set D of previous experiences. It is also used to make efficient use of hardware optimizations and so is essential to learn in mini-batches, rather than online. Such stable

behavior for the algorithm is can be reached setting out the replay buffer large enough to contain a wide range of experiences. The size of the replay buffer is to crucial matter because If it only uses the very-most recent data, you will overfit to that and things will break while if you use too much experience, you may slow down your learning.

Such techniques will be proposed again when describing how the DDPG algorithm (see section 4). In summary, DDPG computes the maximum in continuous action spaces by using a target policy network to compute an action that approximately maximizes $Q_{\phi_{\text{targ}}}$. The target policy network is found by Polyak averaging the policy parameters over the training phase. Finally, the Q-learning in DDPG is accomplished by minimizing the following MSBE loss with stochastic gradient descent:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right],$$

$\mu_{\theta_{targ}}$ *is the target policy..*

The Policy Learning

Policy learning consists to learn a deterministic policy $\mu_\theta(s)$ with which the action that maximizes $Q_\phi(s,a)$ can be received. Assuming the Q-function is differentiable concerning action and taking account of the continuity of actions space is possible to perform gradient ascent to solve:

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi(s, \mu_\theta(s))\right]$$

The algorithm

The base algorithm through which is implemented the DDPG is the following:

FIG. 2. **the algorithm of DDPG**

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_\phi(s,a) - y(r,s',d))^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

    end for
  end if
until convergence

In the next section, it will be shown how the DDPG was implemented from a functional point of view by introducing actor-critic, the Ornstein-Uhlenbeck process and the memory (Replay buffer) used and then showing the pseudo-code of the algorithm.

4. THE SYSTEM IMPLEMENTED ON DDPG FOR CONTINUOUS SPACE

The DDPG implemented for continuous space is based on four neural networks:

- Q-network defined by the parameter: θ^Q
- Deterministic policy function described by the parameter θ^μ
- Target Q network defined by the parameter $\theta^{Q'}$
- Target policy network described by the parameter $\theta^{\mu'}$

How is already told, the application of Q-learning in continuous action spaces is not easy to perform, because the greedy policy needs an optimization action of at every timestep in continuous space; this optimization is too slow to be run with large function approximators and hence an actor-critic architecture has been used on the DPG algorithm. Therefore, for the deterministic policy network and the Q network was implemented as an advantage Actor and Critic architecture and the target networks are time-delayed replicas of original networks that slowly track the learned networks. The DDPG uses target networks because it considerably improves stability in learning avoiding divergence during the update of networks' equations [2]. The pseudo-code of DDPG that was implemented is the following:

FIG. 3. **the algorithm implemented of DDPG**

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

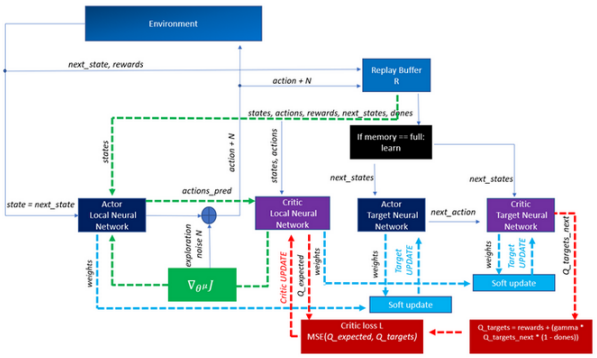
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

The implementation of the algorithm which has been adapted for the project can be divided in the following

step: the use and initialization of Replay buffer, the update of the actor-critic network so policy and value, the update of target networks and the elaboration of the exploration phase characterized by Ornstein-Uhlenbeck Process.

FIG. 4. **DDPG structure**



This steps will be describe in the following sections.

Replay buffer

In order to update the neural network parameters, a memory or so-called replay buffer is used to sample experience. Every the experience's tuples composed by (state, action, reward, the next state) are saved during each trajectory turn-out and memorized to the cache which is the replay buffer. Successively, random mini-batches of experience are sampled from the replay buffer when it needs to update the value and policy networks. The buffer is instantiated though own class which is called during the training of agent.

The actor-critic networks and their update

The actor-critic architecture consist of two entity:

- **Actor** determines which action to take. In DDPG, it is used to approximate the optimal policy deterministically i.e made want always to generate the best believed action for any given state. It follows the policy-based approach, and learns how to act by directly estimating the optimal policy and maximizing reward through gradient ascent.
- **Critic** explains the actor how good its action was and how it should adjust. It utilizes the value-based approach and learns how to estimate the value of different state-action pairs.

The advantage actor-critic is based on idea to decompose the Q-value by the state Value function V(s) and the advantage value A(s, a) and it has been choosen because its advantage function reduces the high variance of policy networks and stabilize the model. Thus, coming back to the application of DDPG, the updated Q value, so the update of critic, is obtained by the Bellman equation:

$$y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})$$

Is important to note that the original Q value is obtained through the value network while the next-state Qvalues are compited with the target value and policy networks. Then the mean-squared loss is minimized between the updated Q value and the original Q value:

$$Lossfunction = \frac{1}{N}\sum_{i=0}(y_i - Q(s_i, a_i|\theta^{Q}))^2$$

The update of actor policy we need to maximize the expected return $J(\theta)$ and calculate the policy loss taking the derivative of the objective function w.r.t the policy parameter:

$$J(\theta) = \mathrm{E}[Q(s,a)|_{s=s_t, a_t=\mu(s_t)}]$$

derivative of $J(\theta)$:

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s,a) \nabla_{\theta^\mu}\mu(s|\theta^\mu)$$

The optimizers use the Adaptive Moment Estimation which computes individual learning rates for different parameters. The actor-critic networks performed have been created setting the following parameters:

Parameters of actor-critic networks	
Parameters	Values
input size	n action (actor), n action + n state (critic)
hidden size	value choosen for hidden size (i.e 256)
output size	n action (actor)

The target networks and their update

The target networks, how it has been already said, are a copy of the actor and critic networks, $Q'(s,a|\theta Q')$ and $\mu'(s|\theta\mu')$ used for calculating the target values. In order to change slowly target values bringing a greatly improving about the stability of learning, the corrispective weights are updated in a soft update manner :

$$\theta^{Q^{'}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q^{'}}$$

$$\theta^{\mu^{'}} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu^{'}}$$

where $\tau \ll 1$

Exploration and Ornstein-Uhlenbeck Process

The exploration is the most important part of the implementation of the algorithm because it allows us to

explore the space in order to obtain good results about learning. In the DDPG case, the space of action is continuous and a performing of exploration results different than its application on discrete space because in this last kind of action space the exploration is done via probabilistically selecting a random action through parameter as epsilon while in continuous space, as in our case, it is performed by adding noise to the action output. During the process of learning both technics were tested and the solution of Ornstein-Uhlenbeck noise reveled the best in this case than the exploration performed by chose random action according to the epsilon parameter (see figure 7). The adding of noise is done using the Ornstein-Uhlenbeck Process which generates temporally correlate exploration for exploration efficiency in physical control problems with inertia and makes a noise that is associated with the previous noise to prevent the noise from canceling out or "freezing" the overall dynamics (for example in this case the DDPG was performed on 'Swimmer' task) [3][2]. Coming back to the algorithm represented in figure 3, the adding of noise through the Ornstein-Uhlenbeck Process appears in the following expression step:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

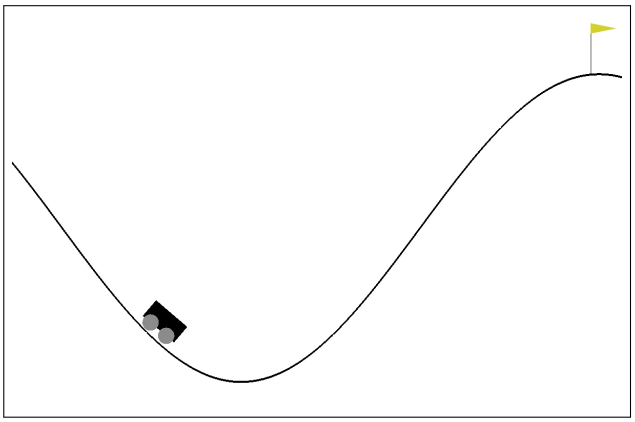
For this part a class was build setting the following parameters:

Parameters of Ornstein-Uhlenbeck process	
Parameters	symbols/value
mu	$\mu/(0,1)$
theta	$\theta/(0,1)$
sigma	$\sigma/(sigmamax, sigmamin)$
decay period	_/100000

5. THE TASK OF TEST'S ENVIRONMENT

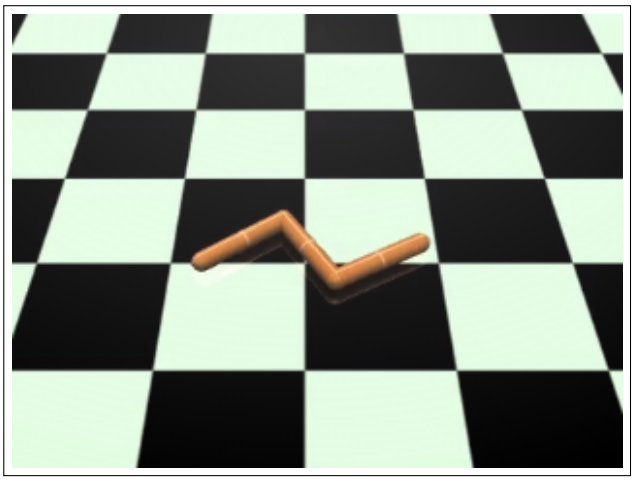
How has already said on introduction, the algorithm was applied in both test and experimental environment. Thus, in this section, the task that concerns the environment of testing will be discussed in order to describe the state space and action space in which the DDPG acted. In this task, a car appears near the bottom of a steep hill and its goal is to actively drive to the top where there is a flag (see figure 5). Since the car is underpowered, it will need to utilize the gas in both forward and reverse directions to roll back and forth numerous times until reaching the goal. It moves under gravity, its own applied acceleration, friction, and being constrained to the curve of the hill. There is a hard inelastic wall on the left side. The applied acceleration can be any continuous value between a positive and negative limit, and the purpose is to reach the top using the least cumulative applied energy. The reward to reach the objective is 100, and otherwise, it is the negative amount of energy applied in each time step due to the applied power. It is important to note that is never any positive reward until the first time reaching the goal, posing an exploration challenge as is shown in the plot of figures 8-9 on page 7-8 in where shows the initial climb of rewards obtained in each episode. An important factor that is joint with the last concept is that the initial random action which is taken by the agent determine determines the initial positive peak if successful or negative vice versa (how in the most case of the example of reward graphs). In Fact, how it will show later, will be the exploration of the key to success for the task.

FIG. 5. **The scenario about of mountain-car-countinous space's task**



The DDPG has been performed also to the experimental environment of 'Swimmer' which aims to make learn at swimming robot with 3-links the practic of swimming on ground characterized by a viscous fluid as a snaker, as is shown in the figure 6 below. In practice, the purpose is to make it swim forward as fast as possible, by actuating the two joints.

FIG. 6. **The scenario about of Swimmer task**



6. THE LEARNING AND RELATIVE RESULTS

The learning in the test environment was carried out initially with several episodes between 100 and 150 with a number of 500 steps. Subsequently, training was carried out up to 200-500-1000 episodes with 1000 steps by changing the parameters concerning the exploration given by the process of Ornstein-Uhlenbeck. In figure 9 is shown the trend of reward and relative average about episodes tested on-mountain car continuous with the best parameters. The important factors about the increase of learning have to focus on the parameter of exploration adopted. In fact, as is reported by figure 8, the increasing of theta from 0.10 to 0.25 brings to better and stable results about learning and this is justified by fact that the agent learns with more exploration.
The best initial values of parameters which allowed to obtain good results about the test environment are reported by the following table:

	the best initial values of parameters	
	parameters	**value**
actor	learning rate	1e-4
	hidden size	256
	optimizer	Adam
critic	learning rate	1e-3
	hidden size	256
	optimizer	Adam
agent	gamma	0.99
	tau	1e-2
	batch size	128
replay buffer	max memory size	50000
noise (OUNoise)	mu	0.1
	theta	0.25
	max sigma	0.3
	min sigma	0.3
	decay period	100000

TABLE I. initial best values for best paramaters

How the graphs show, the reward presents an initial peak that is dulled when is around 10 episodes, then it reaches a good trend with some rapid downs beyond 400 episodes. In order to attenuate these rapid downs reaching stable behavior, the parameter of noise was modified as theta, obtained results that are shown on the final graph.

7. OUTCOMES OF EXPERIMENTAL ENVIRONMENT

For the experimental environment about the swimmer robot, the results are shown in figures 9-10. From the results, it is possible to note that the trend is not stable and not successful than the 'MountainCarContinuous' task. The hyperparameters of adding noise though the Ornstein-Uhlenbeck process, as theta and sigma, have been changed without bringing to a optimal learning but has been noted that an increasing of sigma from its max and min from 0,3 to 0,9 with decreasing of theta from 0,25 to 0,15 have improved general trend of the same, as how the figure 10 show in which there is a several try about a learning of swimmer in 100 episodes. After that, the learning of the swimmer robot has been brought to 1000 episodes and the trend of rewards are shown in figure 11 on page 9 [4]. Finally, similar behavior of learning has been obtained using the Leaky Relu rather than Relu as activation function and the relative results about rewards are shown in figure 12 on page 9.

8. CONCLUSION AND FINAL CONSIDERATIONS

In conclusion, the DDPG algorithm engaged in the elaborate here in obtains satisfactory results as regards the test environment while it obtains slightly unstable results as regards learning in the experimental environment. It must be said that the DDPG turns out to be, as already reported in the various papers that concern it, very sensitive to hyperparameters which involves a deep tuning during the training but at the same time provides for sample-efficient learning. For future work, it is possible to obtain an increase of outcomes from DDPG performing a good further tuning of parameters and change the type of noise process adding to the action during training.

9. REFERENCE

[1] *Reinforcement Learning: An Introduction* Richard S. Sutton, Andrew G. Barto A Bradford Book The MIT Press Cambridge, Massachusetts London, England 2014, 2015

[2] *Continuous control with deep reinforcement learning* countzero, jjhunt, apritzel, heess, etom, tassa, davidsilver, wierstra Google Deepmind London, UK paper at ICLR 2016 –Uhlenbeck Processes and Extensions–Uhlenbeck Processes and Extensions–Uhlenbeck Processes and Extensions

[3] *Ornstein–Uhlenbeck Processes and Extensions* Maller, Ross, Müller, Gernot, Szimayer, Alexander 10.1007/978 − 3 − 540 − 71297 − $8_1$8

[4] *Deep Reinforcement Learning that Matters* Liu, Tianyi,Fang, Shuangsang, Zhao, Yuehui, Wang, Peng, Zhang, Jun McGill University, Montreal, Canada 2018

FIG. 7. **Trend of rewards about 150 episodes obtained with both noise OU process and random action in according to epsilon**

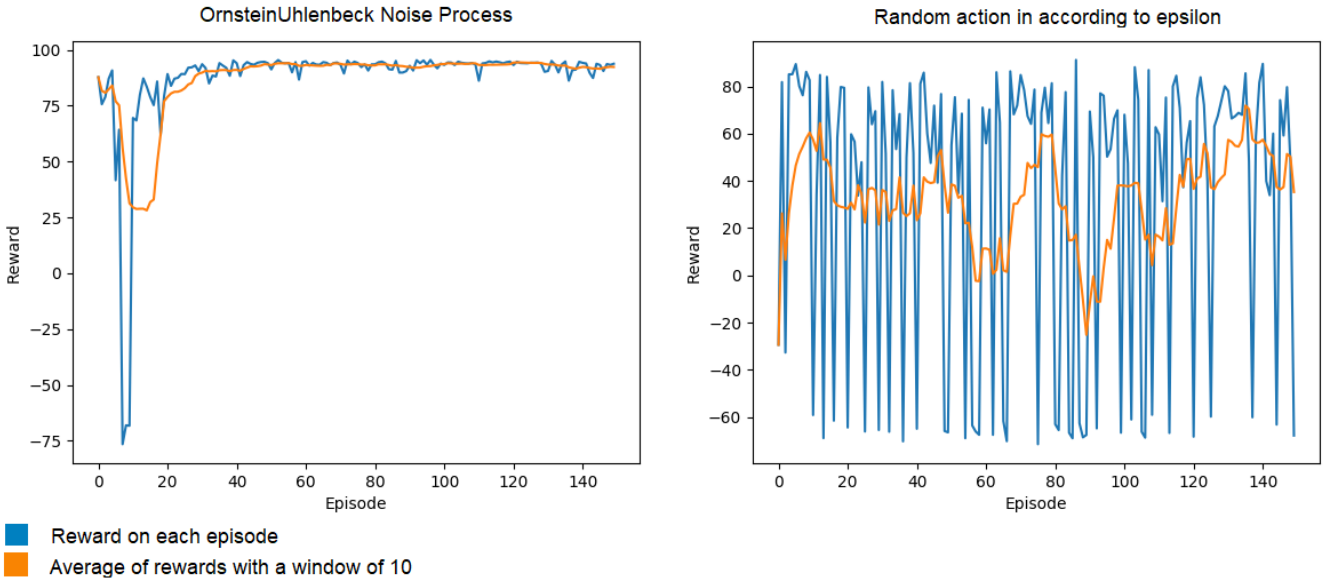


FIG. 8. **The trend of rewards on 1000 episdes with different valuue of theta of adding OUNoise)**

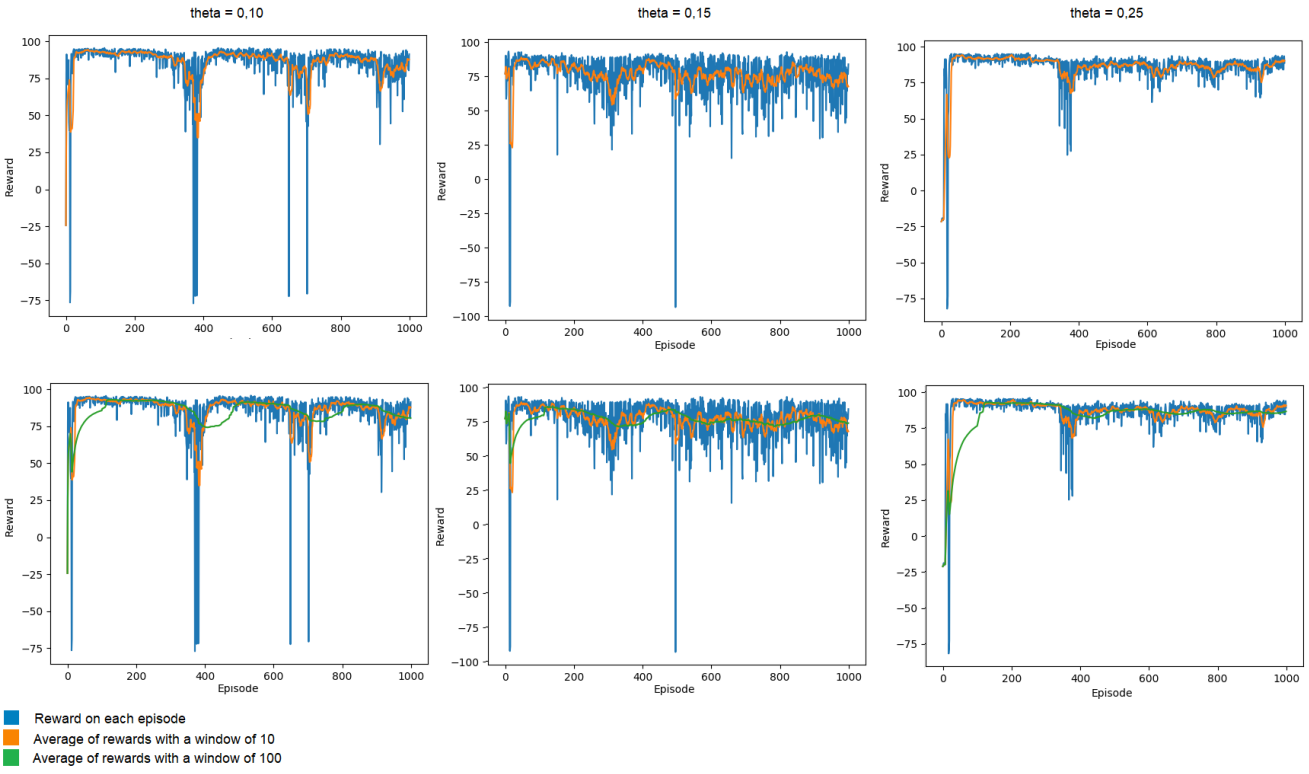


FIG. 9. **The learning of MountainCarContinuous-v0 task from 100 to 1000 episodes with the best hyper-parameters of DDPG found after tuning**

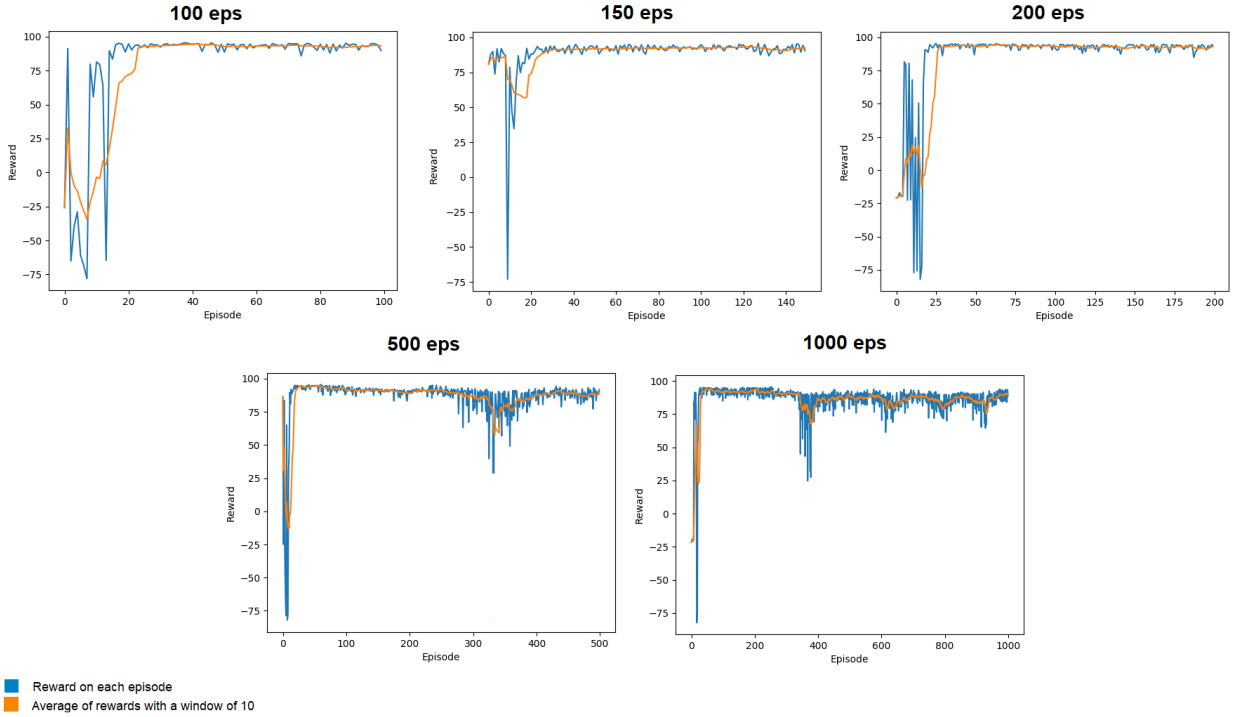


FIG. 10. **The andament of rewards of swimmer robot with 100 episodes perfoming a tuning of parameter of OUNoise theta and sigma.**

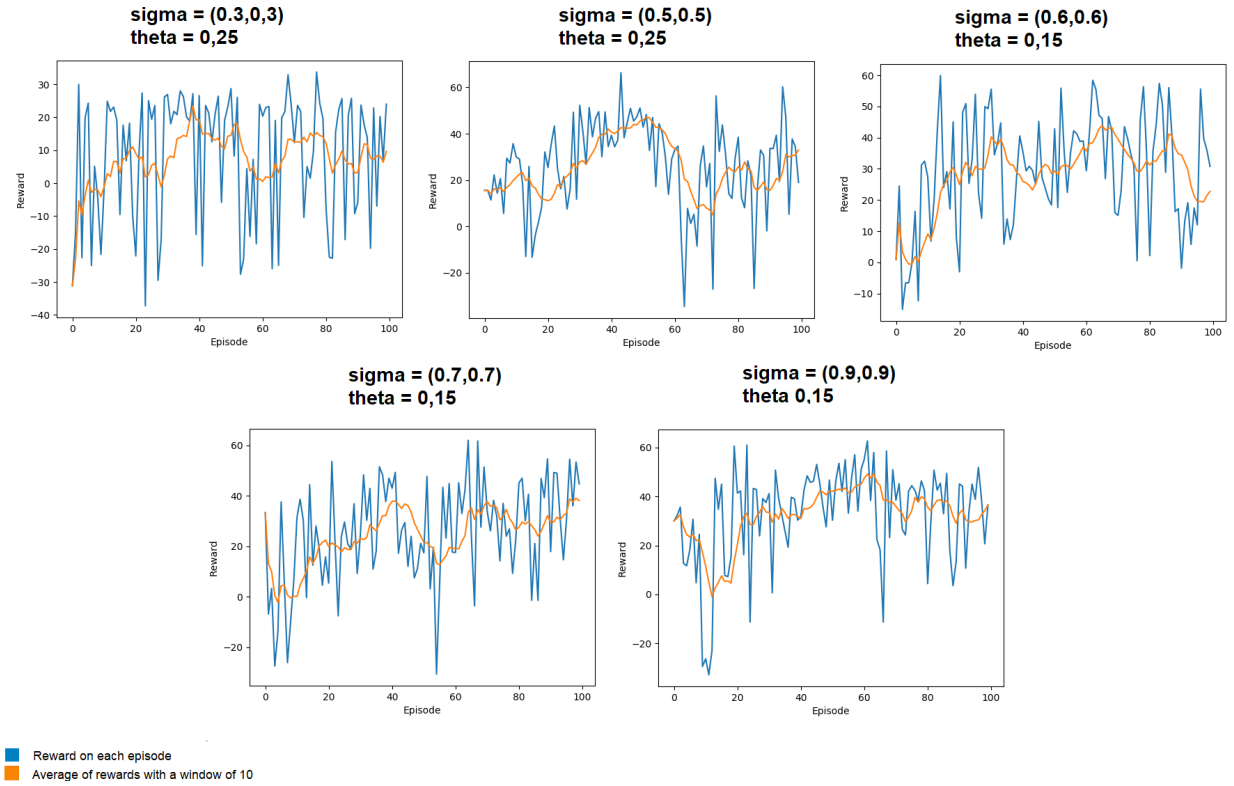


FIG. 11. **The trend of rewards learning of Swimmer trained with 1000 episodes with theta =0.15 and sigma=(0.9,0.9)**

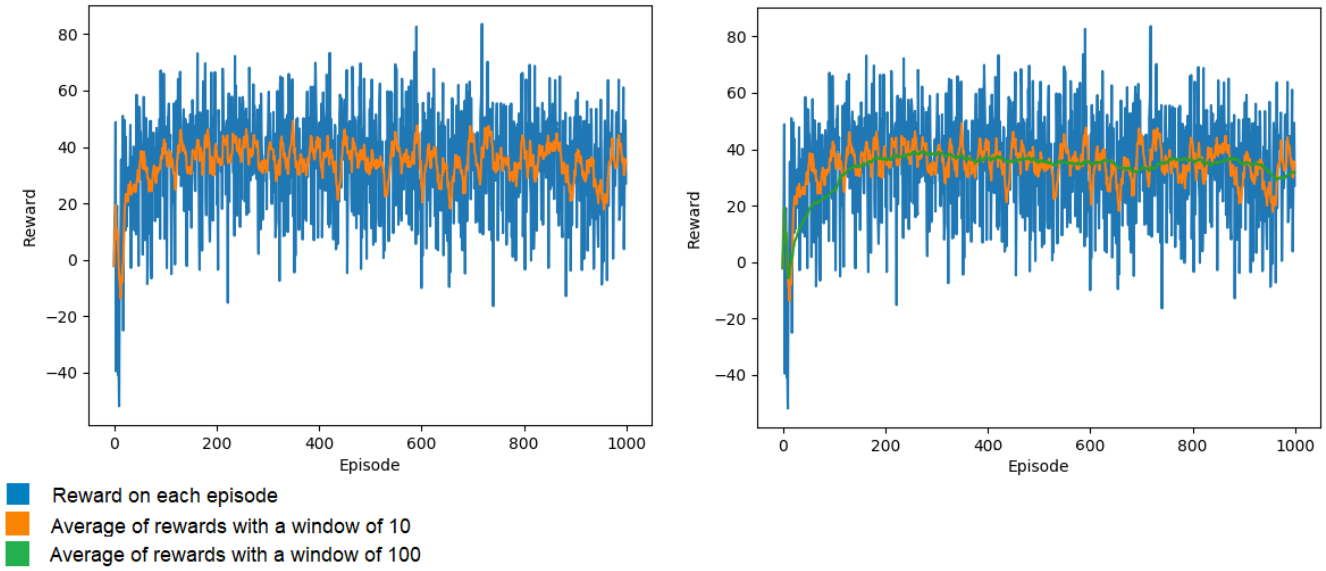


FIG. 12. **The trend of swimmer robot's learning amoung 100-200 episodes with Leaky Relu**