Adan Yusseff Domínguez Ruiz

Deep Reinforcement Learning

21/01/2022

Continuous Deep Reinforcement Learning: Robotic Arm Movement with DDPG

1. Description of the Problem

For this project, a environment was created based on a Unity Environment, the Reacher. It consists in a double-jointed arm that can move to different locations. This is similar to real life scenarios of robotic arms to perform industrial tasks. On this scenario, each step that the agent end-actuator reach the goal location, a reward of +0.1 will be provided.

a. State Space

Can also be called as observation state, consists in 33 variables, corresponding to position, rotation, velocity and angular velocities of the arm.

b. Action Space

Comparing with previous works as tested with DQN architecture, this problem has a continuous action space, meaning that are 4 vectors corresponding to the torque applied on each of the 2 joints from the robotic arm and consisting in a number between <-1, 1>.

c. Rewards

The reward is given each step the end actuator of the arm reaches its goal, in this case, a reward of +0.1 is provided to the agent. In order to solve the scenario, the agent must have a mean reward of +30 after 100 episodes.

2. Algorithm Description

a. Neural Network Architecture

To create the model of the neural network we followed the structure from the first part of the course (Insert reference). In the case of the Deep Deterministic Policy Gradient Algorithm, two neural networks are required, Critic and Actor networks. For the current work, the same architecture was used.

Each network was defined with three fully connected layers, the actor network, using a tangential hyperbolic activation function, and the critic network using only a relu functions. The composition can be seen on Figure 1.

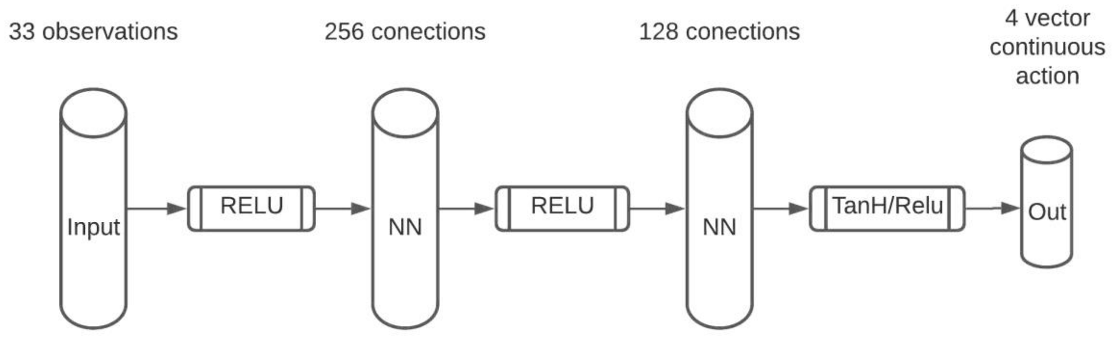

Figure 1: Neural Network architecture, Actor using a TanHyperbolic activation function, while critic uses RELU.

b. Learning Agent

For an artificial intelligence algorithm to work, an agent is required to interact with the environment and gather the information described in the state space, based on it, take an action and select the optimal one giving the maximum reward.

For the agent to get optimal results, some hyper-parameters must be specified. They can be seen on Table 1, however more discussion about the selection of these parameters will be shown on the Results section.

Variable	**Value**
Buffer size of replay buffer	100000
Batch size for sampling	128
Gamma	0.98
TAU	0.001
Learning Rate Actor	0.0001
Learning Rate Critic	0.0001
Weight Decay	0

Table 1: Hyper parameters used for DDPG Agent.

For the agent class, the following functions were defined:

i. Step function

Step function works as the main method for the agent to progress in the learning process. This is where the experience from past episodes are stored in the replay buffer and when the memory is high enough, the agent starts the learning process.

ii. Act function

Act function works as an evaluation method from what the agent has learned. It takes an action based on the current learned information combined with a ground noised to make the agent get a more generalised and robust solution.

iii. Noise Function

Noise in Deep Learning Algorithms helps in the exploration of the actions in the space. The one being used in this project is Ornstein-Uhlenbeck Action space noise, introducing random values inside the action space to unfreeze a stuck learning curve.

iv. Learn Function

How the agent learns is the most important part for deep reinforcement learning, a neural network needs to learn to get the right weights to produce the expected and optimal result. In the case of the current project, deep deterministic policy gradient algorithm is used (DDPG).

v. Update Function

This function is used to update the values of the neural networks.

vi. Replay Buffer

Replay buffer is used in DDPG to store values from previous experiences and use it in the learning process (Sutton and Barto, 1998).

vii. Deep Deterministic Policy Gradient

On previous tasks we used Deep Q-Network as a Deep Reinforcement Learning algorithm to learn solve a specific task over a discrete action space. For continuous action spaces, this technique does not work as intended. DDPG born as a combination of DQN, using the Experience Relay and slow-learning targets, and Deterministic Policy Gradient using the capabilities to work over continuous spaces. Contrasting with previous methods, this algorithm uses two target neural networks, an Actor Neural network to propose an action given a state and a Critic neural network to predict the policy and the performance of an action given a state and an action (Lillicrap *et al.*, 2019).

viii. Training Process

To train the neural network, a set of steps are required:

1. Loop for the number of maximum Episodes

    i. Initialize scores based on the number of agents.

    j. Reset the agent to reset the noise.

    k. Get an action based on the observable state

    l. Get next state and reward based on the action found.

    m. Save the data on the replay buffer and train the NN's based on the state, next state and reward values.

n. Update the weights on NN's

o. If mean of reward is more than 30, the agent is trained.

3. Discussions and Results

By using the DDPG algorithm as based with no changes, the agent was not able to learn and obtain more than +1.0 cumulative reward, so a few tweaks were necessary.

As a recommendation from the course, it was decided to modify the actor/critic neural networks to add a normalization layer between each fully connected layer.

Another change was performed, using a similar approach than the one used in discrete action space DQN. We modified how many episodes needed to pass before updating the neural network, it was decided to update every 10 episodes.

Results looked promising during the first episodes, however, as seen on Figure 2, the learning process got stuck and even went getting smaller rewards.

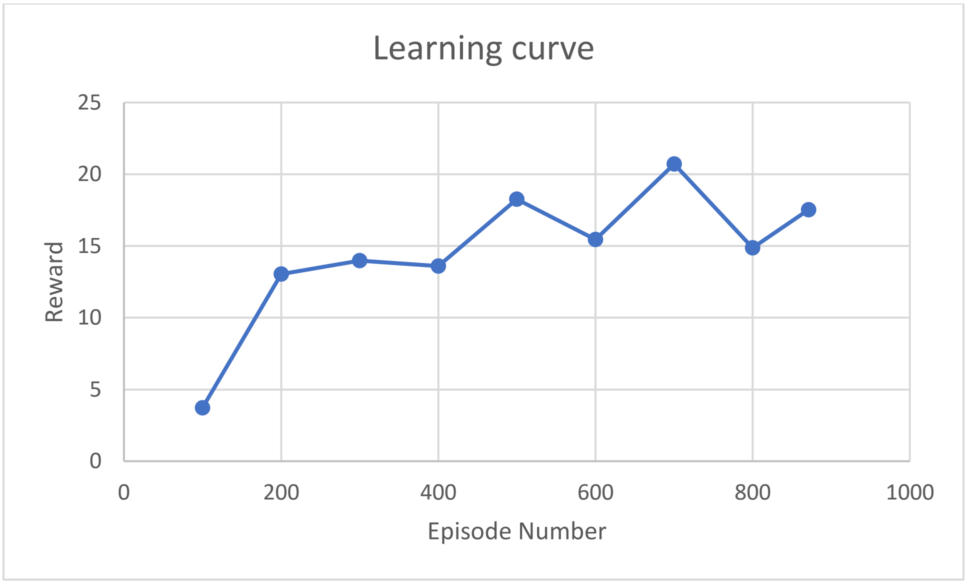

Figure 2: Learning curve using NN model with normalization layers and update them every 10 episodes.

Other changes were decided to be done, normalization layers were removed for the neural network architecture and the update process occurred every episode. With these changes and the hyper parameters described in Table 1, the learning curve was better, and the solution came in just 210 episodes as shown in Figure 3.

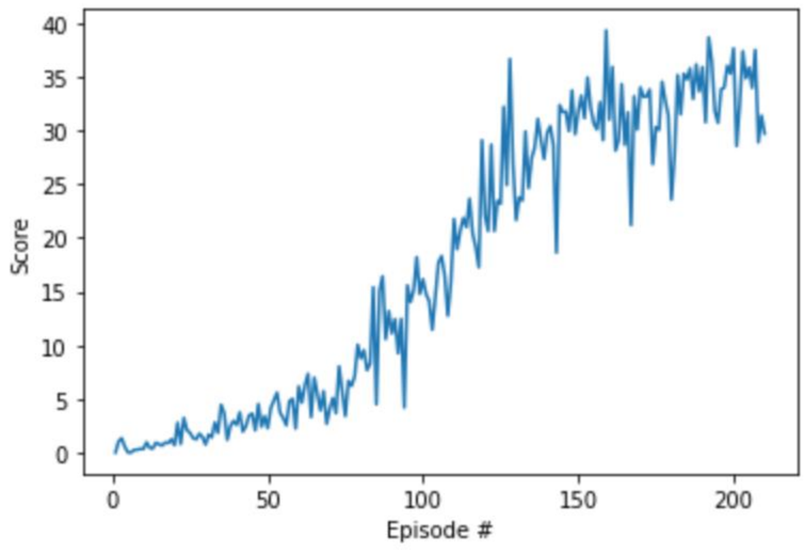

Figure 3:Solution with the updated algorithm DDPG.

With the gathered information and results it was cleared that DDPG is a very powerful algorithm for continuous tasks, however, it falls into an issue of finding the best hyper parameters to be used for different environments. Since there is not a defined method to get a "correct" parameter, a trial-and-error method is used to get what could give an optimal learning curve to get the solution in less episodes.

4. Future Work

In the present project we found different issues with the action space noise, since even with it, the model got stuck many times. It would be an interest next step to research and apply a parameter space noise, since it could potentially help in less trial-and-error hyperparameter finding and better resolution time(Colas, Sigaud and Oudeyer, 2018).

Another option for the resolution of this environment will be the appliance of the Distributed Distributional Deterministic Policy Gradient (D4PG), which seems to optimize the learning process of the DDPG by using N-Step returns and prioritized experience replay (Barth-Maron *et al.*, 2018).

Works Cited

Barth-Maron, G. *et al.* (2018) 'Distributed Distributional Deterministic Policy Gradients', *arXiv:1804.08617 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1804.08617 (Accessed: 24 January 2022).

Colas, C., Sigaud, O. and Oudeyer, P.-Y. (2018) 'GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms', *arXiv:1802.05054 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1802.05054 (Accessed: 24 January 2022).

Lillicrap, T.P. *et al.* (2019) 'Continuous control with deep reinforcement learning', *arXiv:1509.02971 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1509.02971 (Accessed: 24 January 2022).

Sutton, R.S. and Barto, A.G. (1998) *Reinforcement learning: an introduction*. Cambridge, Mass: MIT Press (Adaptive computation and machine learning).