Adan Yusseff Domínguez Ruiz

Deep Reinforcement Learning

10/02/2022

Multi-Agent Deep Reinforcement Learning: Tennis competitive play with DDPG

1. Description of the problem
    a. State Space

For the current project an environment was created by the Udemy experts based on the Tennis competitive play from Unity Environment. Similar at when two human players play between each other and get better over time by learning from their mistakes. On this scenario, two rackets need to bounce a ball over a net. If done right, it will receive a reward of +0.1. If an agent let the ball out of bounds or to hit the ground, it receives a reward of -0.01. The goal is to keep the ball in play, obtaining an accumulative reward of +0.5.

    b. Action Space

Consists of the two movements by each racket, defined in one for the movement toward or away the net and the second one for jumping. Both actions are in continuous state.

    c. Observation State

Observation state is independent by each agent, consisting in the position and velocity of the ball and racket.

2. Learning Algorithm
    a. Neural Network

The neural Network used in the current project was the same used in the "Continuous Learning Reacher" from the 2nd Project (Dominguez, 2022). For the Deep Deterministic Policy Gradient, 2 neural networks were required, one for the Actor and one for the Critic.

For the critic neural network, three fully connected layers were used, using RELU activation functions in between as shown in Figure 1. For the actor neural network, a TanHyperbolic activation function was used, however, a normalization layer was used in between each fully connected layer.

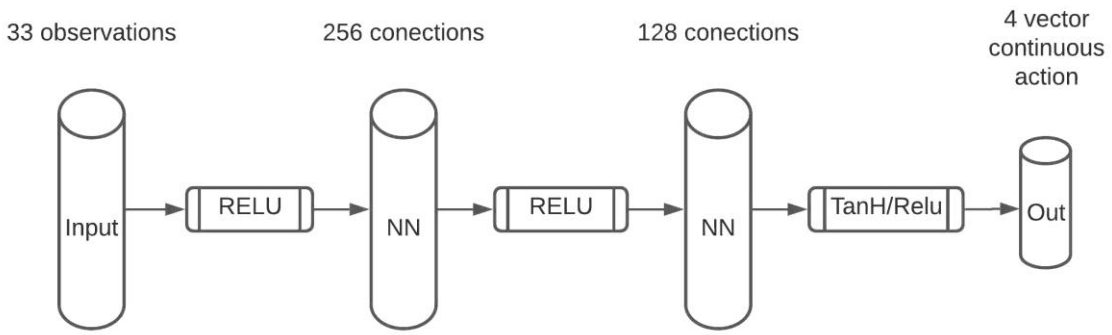

Figure 1: Neural Network architecture, Critic using RELU activation functions and TanH for the Actor.

b. Learning Agent

In this environment, two learning agents were required, so it was important two create a new class to manage any number of existing agents in the environment. For this it was important to create new functions on a multiagent class.

i. Step Function

Step function works as with one single agent except for using the learning process in a loop to update them individually.

ii. Act Function

Act function works as an intermediary to send the actions from the environment into each single agent existing. Then, based on the current knowledge, combined with some noise, gives a more generalized and robust solution.

iii. Deep Deterministic Policy Gradient (DDPG)

DDPG was created 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. The algorithm uses two target neural networks, an actor to propose an action given a state and a critic to predict the policy and the performance of an action given a state and an action (Lillicrap *et al.*, 2019).

For the algorithm to properly work, a series of hyperparameters is required, these are shown in Table 1.

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: Hyperparameters used in the ddpg_agent class

iv. Training Process

To train the neural networks, a set of steps are required, however, it is important to, instead of initializing a single agent, to create a loop to initialize the two different agents in the environment.

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 +0.5 over 100 episodes, the agent is trained.

3. Discussions and Results

The development of the current project started as a training with the DDPG agent developed in the 2nd project, however, the states and actions gathered from a multiagent environment were not adapted to the developed agent class. The new multi-agent class needed to be developed basically with internal "for" loops for each agent to act individually with many observations and actions.

First attempt started took a long time and had instability at many episodes, increasing the rewards and then failing at the process. A normalization batch layer was added into each layer of the actor neural network. It was found that with this modification and increasing the number of hidden layers, the process got faster.

With the described changes, we got a solution in less than 800 episodes, however, the results were still not stable, and even when the mean reward was +0.5, when testing it, this was still failing randomly.

```
Episode 100     Average Score: -0.00
Episode 200     Average Score: 0.010
Episode 300     Average Score: 0.02
Episode 400     Average Score: 0.05
Episode 500     Average Score: 0.10
Episode 600     Average Score: 0.18
Episode 700     Average Score: 0.32
Episode 800     Average Score: 0.57

Environment solved in 800 episodes!     Average Score: 0.57
```

Figure 2: Episodes and Results.

To solve this issue, the verification for the reward was done after each 100 episodes, with this modification, the agents behaved properly after enough training.

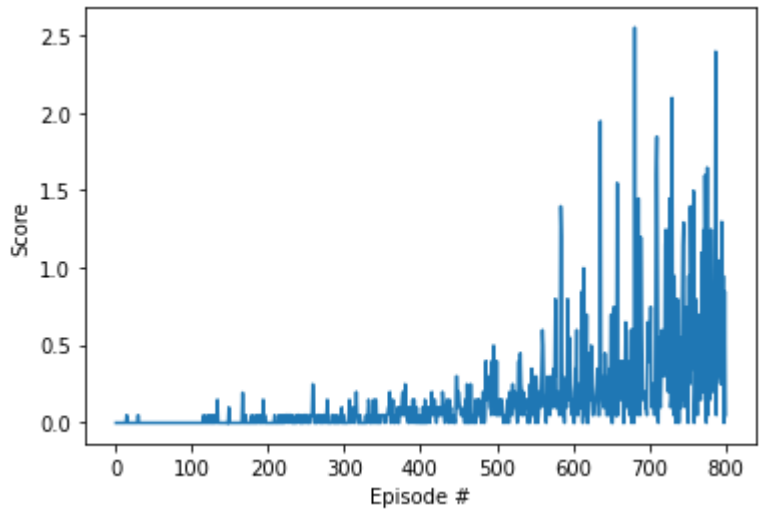

Figure 3: Scores learning graph.

After the 800 episodes, it was possible to see that the model was stable in each episode, the rackets moving at an accurate rate to keep the ball on play as seen on Figure 4.

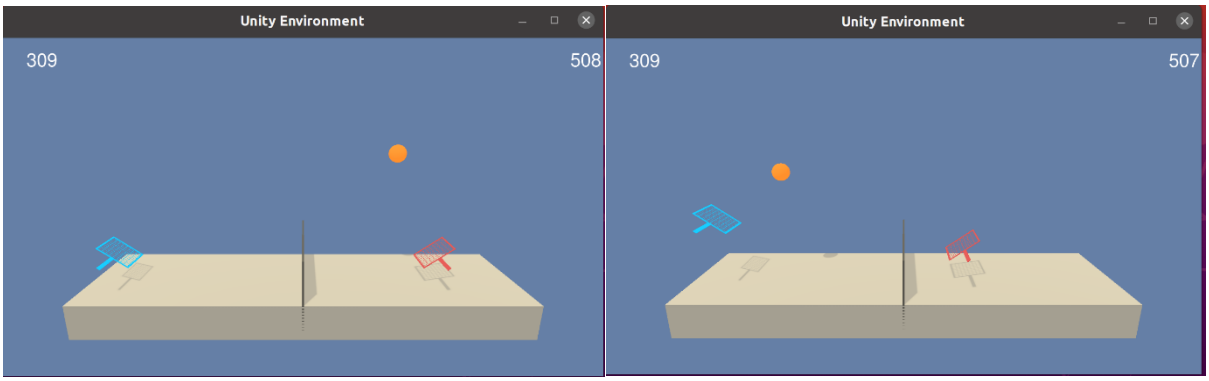

Figure 4: Final model tested.

4. Future Work

In the future it will be expected to attempt other algorithms to adapt in multi-agent systems. This was a competitive scenario, where only two agents were in play with independent observations.

It will be an interesting next step to perform the learning process in a shared space, using the same observations for a common objective. The football environment could potentially be a good learning task.

References

Dominguez, A. (2022) *ContinuousLearning_Reacher*. Available at: https://github.com/YusseffRuiz/ContinuousLearning_Reacher (Accessed: 10 February 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).