Reacher continuous space problem

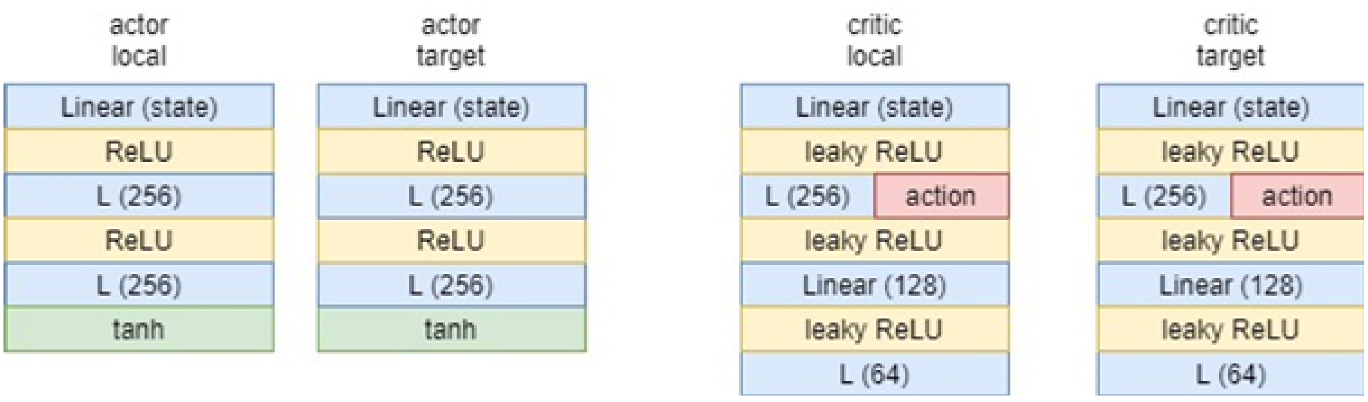
Guillermo del Valle Reboul

1 Algorithm Used

The **Deep Deterministic Policy Gradient (DDPG),** a model-free off-policy algorithm for learning continuous actions, was used for this project based in a Unity Reacher environment. It combines ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). It uses Experience Replay and slow-learning target networks from DQN, and it is based on DPG, which can operate over continuous action spaces.

The Actor Critic model is based in simple and relatively shallow neural networks.

- For the local and target ***Actor Neural Network***, 3 Linear layers fully connected are used with sizes “state_size”, 256 and 256 and Relu activation functions.
- For the other neural networks of the ***Critic Neural Network***, a little bit more complex architecture is required to achieve a good convergence with 4 layers. The sizes are “state_size”, 256 with the action attached, 128 and 64, with no output activation function in the last layer. The use of leaky ReLU proved to improve performance for the critic, but the main responsible of the achievement of an average reward of 30 is the 4-layer NN structure and the learning rate of 1e-4.



2 Hyperparameters

```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-4        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
EPSILON = 1.0
EPSILON_DECAY = 1e-6
LEARN_EVERY = 10
```

3 Files Used

▫ Continuous_Control.ipynb: used in training.

▫ continuous_control_DDPG.py: script that fulfils the same purpose than the notebook.

▫ ddpg_model.py: contains the DDPG Network model used for both target and local networks in actor-critic architecture.

▫ ddpg_actor_critic_model.py: contains the agent with the DDPG algorithm.

▫ checkpoint_actor.pth: weights saved from training the actor.

▫ checkpoint_critic.pth: weights saved from training the critic.

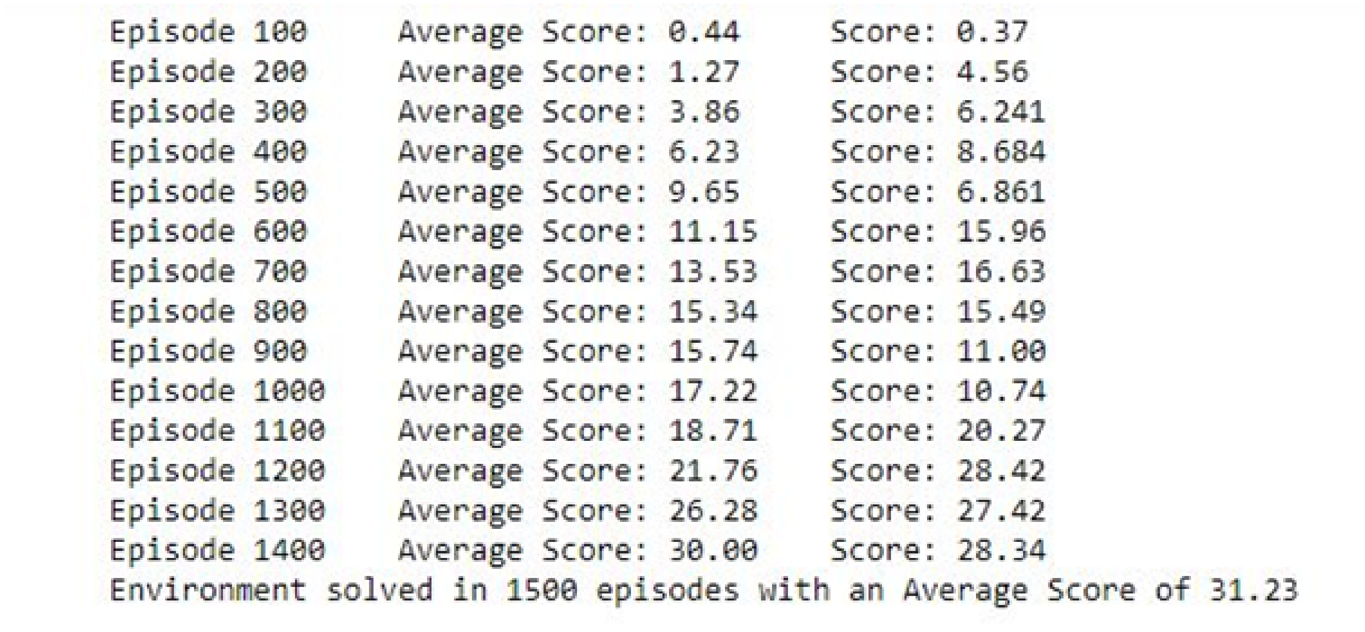
▫ README.md: for instructions.

▫ media: folder that contains video of the agent's performance.

4 Plot of Rewards

DDPG Solved the environment in 1500 episodes (Average score = 31.23).

See "/media/trained_speedup.mp4" to watch trained agent in action.



```
Episode 100     Average Score: 0.44     Score: 0.37
Episode 200     Average Score: 1.27     Score: 4.56
Episode 300     Average Score: 3.86     Score: 6.241
Episode 400     Average Score: 6.23     Score: 8.684
Episode 500     Average Score: 9.65     Score: 6.861
Episode 600     Average Score: 11.15    Score: 15.96
Episode 700     Average Score: 13.53    Score: 16.63
Episode 800     Average Score: 15.34    Score: 15.49
Episode 900     Average Score: 15.74    Score: 11.00
Episode 1000    Average Score: 17.22    Score: 10.74
Episode 1100    Average Score: 18.71    Score: 20.27
Episode 1200    Average Score: 21.76    Score: 28.42
Episode 1300    Average Score: 26.28    Score: 27.42
Episode 1400    Average Score: 30.00    Score: 28.34
Environment solved in 1500 episodes with an Average Score of 31.23
```

Figure 4-1. Summary of the agent's performance.

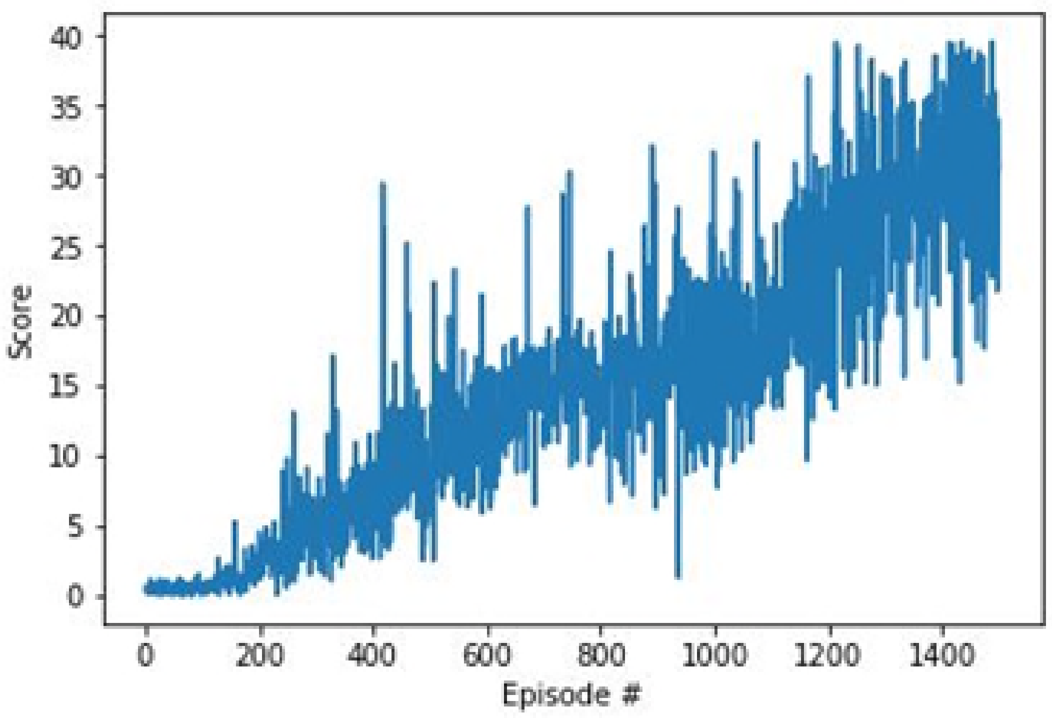


Figure 4-2. Plot of the agent's performance over the 1500 episodes.

5 Lessons learnt

- The Critic Network used a **4-layer neural network** with a relatively low learning rate (1e-4) which proved to be the *key element* that made the agent learn faster and achieve high rewards.
- The addition of Ornstein-Uhlenbeck Noise proved to be a little bit problematic, and for that reason the random.random noise generator was replaced by a standard random noise. This addition of noise to the action was important for the learning process. Also, I applied epsilon decay.
- The use of an update every 10 steps in the agent's neural networks proved to be important for achieving convergence. Every 10 steps, the actor and critic networks would pick data from the memory buffer to learn and update their weights.
- The use of different batch sizes (64, 128, 256) did not seem to be relevant for achieving converge.

6 Ideas for Future implementations

- The addition of Batch Normalization was contemplated in my implementation for some time, but I did not detect an increase in the performance. Surely, it needs further investigation into the matter since, theoretically, batch norm has been proved to improve learning and convergence.
- The use of another algorithm like D4PG can be explored as well as PPO, A2C, A3C for the multiple agent's example.