

REPORT OF IMPLEMENTATION

In this project was used Actor-Critic method, through the DDPG algorithm. DDPG means Deep Deterministic Policy Gradient and extend policy-based reinforcement learning methods to complex problems using deep neural networks.

DDPG is very similar to DQN method, using ReplayBuffer. A big difference is that DQN is used to discrete action spaces and DDPG is used to continuous action spaces.

In algorithms like DDPG that implement actor-critic method we have two neural networks, one is the actor and another is the critic.

Also, was used Experience Replay and Fixed Q-Targets methods to improve the agent's performance.

CODE

1- model.py

In this file was created the actor and critic classes, each one with your own neural network. The neural network was created using PyTorch.

2- ddpg_agent.py

In this file was created the Agent class.

Looking for learning and improving the results, the agent implements the Experience Replay and Fixed Q-Targets methods.

Was created the ReplayBuffer class to implement the Experience Replay method

4- Training the agent

The agent was trained in 2.000 episodes and epsilon decay = 0.995

The environment is considered solved when the agent achieve the mean score ≥ 13.0 , but the train continues until finish the 2.000 episodes.

ATTEMPT 1

Model.py

Actor model architecture:

- three fully connected layers, receiving the state as input and actions as output.
- 33(Input-state_size) x 400(hidden_layer) x 300(hidden_layer) x 4(output - action_size)
- Was applied ReLu activation at hidden layers and Tanh activation at output layer.

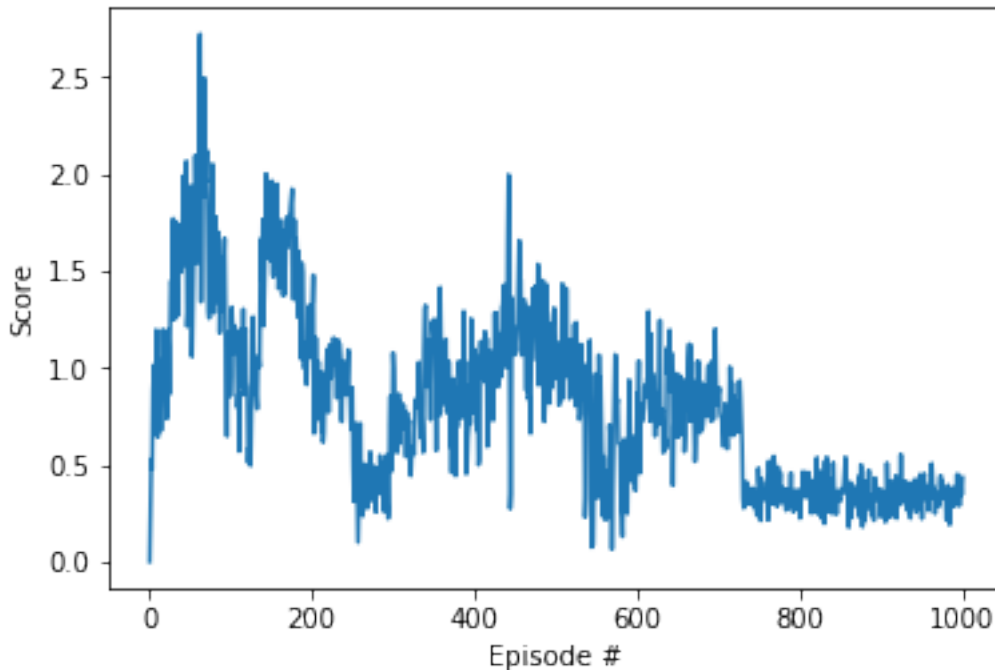
Critic model architecture:

- three fully connected layers, receiving the state as input and return as output only one unit, the Q-values.
- 33(Input-state_size) x 404(hidden_layer+action_size) x 300(hidden_layer) x 1(output - Q-values)
- Was applied ReLu activation at hidden layers.

Hyperparameters used to train the agent:

```
BUFFER_SIZE = int(1e5) # replay buffer size
BATCH_SIZE = 128      # 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-3       # learning rate of the critic
WEIGHT_DECAY = 0       # L2 weight decay
UPDATE_EVERY = 1       # how often to update the network
N_EPISODES = 1000      # number of episodes
```

The result of the first attempt was really bad. The max scored achieved was around 2.5



ATTEMPT 2

To second attempt I did some changes:

- Neural network architecture of the Actor
- Neural network architecture of the Critic
- Hyperparameters
- UoNoise

Model.py

Actor model architecture:

- three fully connected layers, receiving the state as input and actions as output.
- 33(Input-state_size) x 256(hidden_layer) x 128(hidden_layer) x 4(output - action_size)
- Was applied ReLu activation at hidden layers and Tanh activation at output layer.

Critic model architecture:

- four fully connected layers, receiving the state as input and return as output only one unit, the Q-values.
- 33(Input-state_size) x 132(hidden_layer+action_size) x 64(hidden_layer) x 32(hidden_layer) x 1(output - Q-values)
- Was applied ReLu activation at hidden layers.

Hyperparameters used to train the agent (Only modified hyperparameters):

```
BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 1024      # minibatch size
```

"""Ornstein-Uhlenbeck process."""

Was necessary to change the noise process. Below the old and new version of code.

```
#dx = self.theta * (self.mu - x) + self.sigma * np.array([random.random() for i in range(len(x))])  
dx = self.theta * (self.mu - x) + self.sigma * np.random.standard_normal(self.size)
```

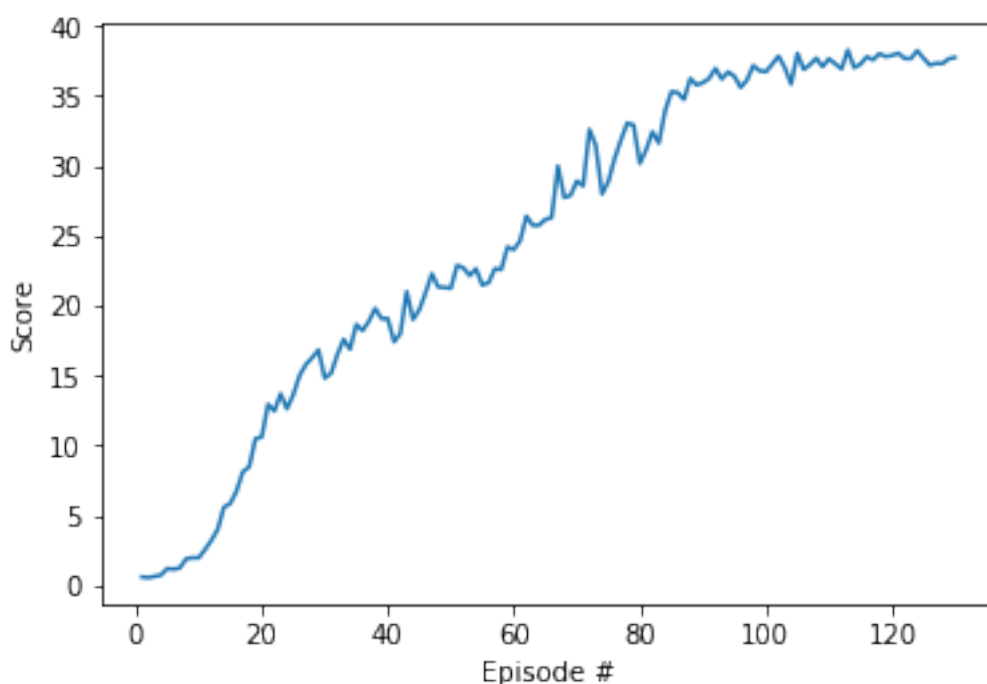
RESULT

With those architecture and hyper parameters was possible achieve the Average Score of the 30.19 in only 130 episodes.

Below the progress of the agent's learning.

Episode 10	Average Score: 1.19	Episode Score: 1.96
Episode 20	Average Score: 3.87	Episode Score: 10.60
Episode 30	Average Score: 7.39	Episode Score: 14.82
Episode 40	Average Score: 10.04	Episode Score: 19.07
Episode 50	Average Score: 12.07	Episode Score: 21.24
Episode 60	Average Score: 13.84	Episode Score: 24.01
Episode 70	Average Score: 15.72	Episode Score: 28.91
Episode 80	Average Score: 17.60	Episode Score: 30.17
Episode 90	Average Score: 19.45	Episode Score: 35.95
Episode 100	Average Score: 21.16	Episode Score: 36.73
Episode 110	Average Score: 24.76	Episode Score: 37.66
Episode 120	Average Score: 27.87	Episode Score: 37.90
Episode 130	Average Score: 30.19	Episode Score: 37.76

Environment solved in 130 episodes



Future ideas for improving agent's performance

Looking for improving the agent's performance we suggest that be implemented another actor-critic algorithms like PPO and A3C to compare the results with DDPG.