Project Report

This report highlights the Actor and Critic networks with Deep Deterministic Policy Gradients learning agent[i] along with the hyperparameters and provides some references for future work.[ii]

Actor Network

The network is a feed-forward network with 3 fully connected ReLu activation layers as follows

1. Input 33 (see state size) => Output 400
2. Input 400 => Output 300
3. Input 300 => Output 4 (action size)

with tanh activation function that maps state -> action values.

Critic Network

The network is a feed-forward network with 4 fully connected leaky ReLu activation layers as follows

1. Input 33 (see state size) => Output 256
2. Input 256 => Output 256
3. Input 256 => Output 128
4. Input 128 => Output 1

Learning Agent

The learning agent is created as a class that interacts and learns from the environment with local/target actor and critic network as initialized variants plus Adam variant as optimizer.

Hyperparameters

- Replay buffer size => 1e5
- Minibatch size => 128
- Discount factor Gamma => 0.99
- TAU for soft update of target parameters => 1e-3
- Learning rate (Actor) => 1e-4
- Learning rate (Critic) => 2e-3
- Weight decay => 0.0001

Training

The training of the agent via ddpg function (see Continuous_Control.ipynb) is running the following steps in every episode:

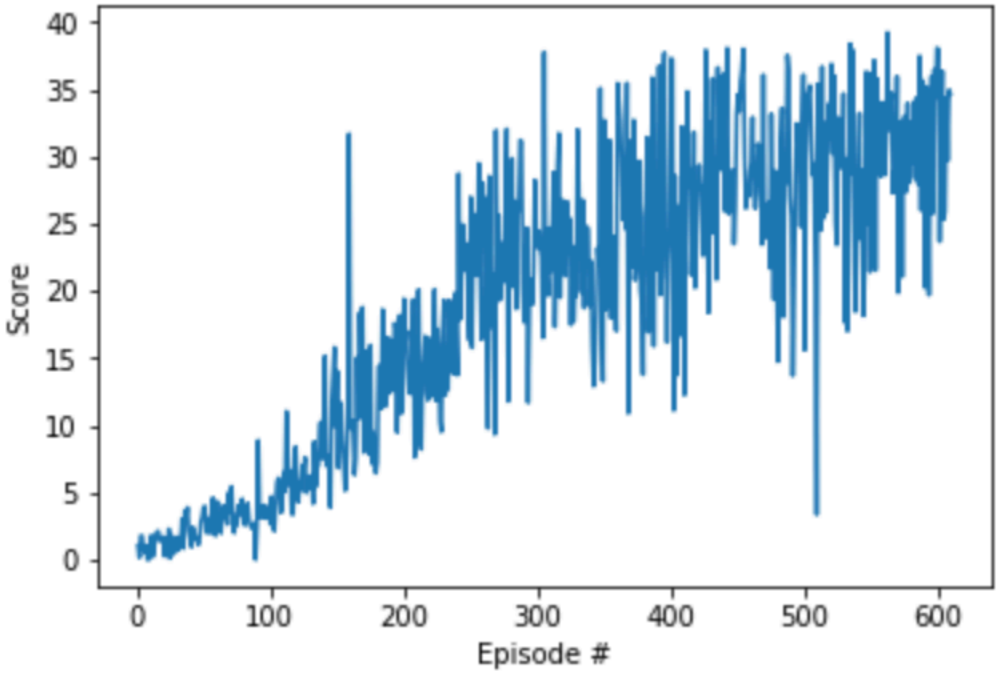
- Return actions for given state as per current policy from actor/critic local network

- Save experience in replay memory
- Learn every defined update time steps if enough samples are available in memory with random subset to update value parameters using given batch of experience tuples
    1. Get max predicted Q values (for next states) from actor/critic target network
    2. Compute Q targets for current states
    3. Get expected Q values from actor/critic local network
    4. Compute and minimize MSE loss
    5. Soft update target network

In total it is designed to run over 2,000 episodes (with 1,000 iterations per episode) - but it is considered as solved and hence ends if the agent gets an average score of +30 over 100 consecutive episodes.

Plot of Rewards

```
Episode 25      Average Score: 1.095    Score: 0.100
Episode 50      Average Score: 1.512    Score: 3.880
Episode 75      Average Score: 2.045    Score: 2.550
Episode 100     Average Score: 2.369    Score: 2.640
Episode 125     Average Score: 3.481    Score: 6.7200
Episode 150     Average Score: 5.127    Score: 14.030
Episode 175     Average Score: 7.248    Score: 15.950
Episode 200     Average Score: 9.712    Score: 19.400
Episode 225     Average Score: 11.839   Score: 17.150
Episode 250     Average Score: 14.217   Score: 26.980
Episode 275     Average Score: 16.768   Score: 20.730
Episode 300     Average Score: 19.248   Score: 23.210
Episode 325     Average Score: 21.664   Score: 17.470
Episode 350     Average Score: 22.689   Score: 32.690
Episode 375     Average Score: 23.552   Score: 21.950
Episode 400     Average Score: 24.105   Score: 37.290
Episode 425     Average Score: 24.456   Score: 26.800
Episode 450     Average Score: 26.390   Score: 34.600
Episode 475     Average Score: 27.398   Score: 33.160
Episode 500     Average Score: 27.755   Score: 15.590
Episode 525     Average Score: 28.852   Score: 32.890
Episode 550     Average Score: 28.292   Score: 30.460
Episode 575     Average Score: 28.635   Score: 33.020
Episode 600     Average Score: 29.733   Score: 37.970
Episode 609     Average Score: 30.036   Score: 34.640
Environment solved in 509 episodes!     Average Score: 30.036
```



Future Work

For training purposes, noise adding was paused since the learning process broke down quite early after only a few dozens of episodes – no matter how the different hyperparameters were tuned. In next stage the batch normalization should be added to the respective network architectures to make the learning more robust against added noise.

Additional enhancements can be achieved via implementing some benchmarking from Yan Duan, Xi Chen, Houthooft R., Schulman J., Abbeel P., Benchmarking Deep Reinforcement Learning for Continuous Control, arXiv:1509.02971.

---

[i] Implementations of the Deep Deterministic Policy Gradients adapted from https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum and https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-bipedal

[ii] Lillicrap et. al., Continuous control with deep reinforcement learning (2015), https://arxiv.org/abs/1509.02971