Project-2 : Deep Reinforcement learning - Continuous Control

DEEP DETERMINISTIC POLICY GRADIENT (DDPG) Algorithm

About the Algorithm:

Algorithm used in the project is further modified from the vanilla DDPG agent in solving single agent pendulum environment.

DPG is a policy based learning algorithm in which the agent will learn from the non-processed observation spaces without knowing the environment. In contrast to a DQN which learns directly through a gradient method which estimates the weights of the optimal policy.

DDPG is a model-free policy based learning algorithm in which the agent will learn directly from the un-processed observation spaces without knowing the domain dynamic information. That means the same algorithm can be applied across domains which is a huge step forward comparing with the traditional planning algorithm.

In contrast with DQN that learn indirectly through Q-values tables, DDPG learns directly from the observation spaces through policy gradient method which estimates the weights of an optimal policy through gradient ascent which is similar to gradient descent used in neural network. Also, policy based method is better suited in solving continuous action space environment.

DDPG deploys an Actor-Critic model in which the Critic model learns the state-value function and uses this to determine how the Actor's policy model should change. The Actor learns from the continuous space without the needs for many data samples since it can rely on the critic to give it feedback on good and bad actions.

However, stability could be a problem with this approach and to mitigate the challenge of unstable learning, there are several techniques that can be employed like, Gradient Clipping, Soft Target Update, Twin local/ target networks, and Replay Buffer. Reply Buffer is most critical as it allows the Deep Deterministic Policy Gradient (DDPG) agent to learn offline by gathering experiences collected from the environment agents and sample experiences from a large memory buffer across experiences.

Model Architecture

The model built using DDPG code in PyTorch was used and adapted for this 20 agent environment.

The Actor model is a neural network with 2 hidden layers,

- Hidden: (input, 128) - ReLU
- Hidden: (128, 128) - ReLU
- Output: (128, 4) - TanH

The Critic model is similar to Actor model except the final layer is a fully connected layer that maps states and actions to Q-values.

- Hidden: (input, 128) - ReLU
- Hidden: (128 + action_size, 128) - ReLU
- Output: (128, 1) - Linear

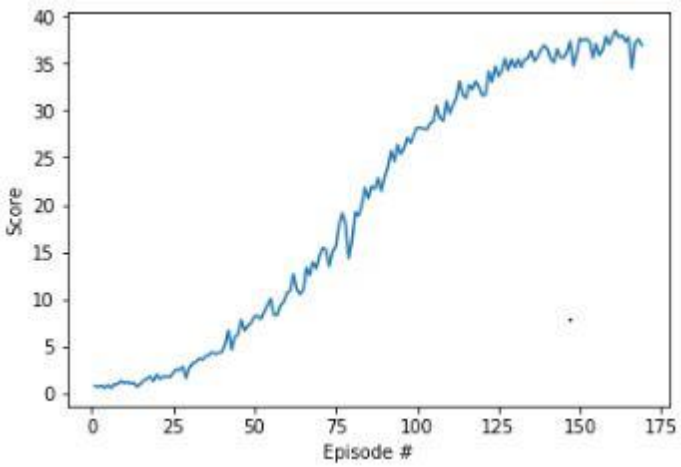
Hyperparameters

- BUFFER_SIZE = int(1e6) # 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-4 # learning rate of the critic
- WEIGHT_DECAY = 0.0 # L2 weight decay

Training Results:

The figure below show the performance and results of the model. The model performs well. The DDPG agent takes 169 episodes to achieve the goal.

```
Episode 164     Total Average Score: 29.02      Mean: 37.28     Min: 34.44      Max: 39.07      Duration: 15.79
Episode 165     Total Average Score: 29.29      Mean: 37.75     Min: 32.75      Max: 39.47      Duration: 15.54
Episode 166     Total Average Score: 29.50      Mean: 34.45     Min: 28.86      Max: 37.85      Duration: 15.83
Episode 167     Total Average Score: 29.75      Mean: 37.14     Min: 33.97      Max: 39.44      Duration: 16.21
Episode 168     Total Average Score: 29.98      Mean: 37.56     Min: 36.11      Max: 39.32      Duration: 16.01
Episode 169     Total Average Score: 30.22      Mean: 36.90     Min: 34.74      Max: 39.02      Duration: 16.27
Problem Solved after 169 epsisodes!! Total Average score: 30.22
```



Future Improvements

There could be further improvement in applied DDPG algorithm by applying Priority Experienced Replay in which important experience will be sampled more often. Based on the paper "A novel DDPG method with prioritized experience replay", This will further reduce the training time, improve the stability of the training process and is less sensitive to the change in hyperparameters.