Continuous Control Project

Introduction

The aim of this project was to implement and train an agent/s to successfully move a double-jointed arm to target locations in a virtual world. This is an episodic problem with the goal of maintaining its position at the target location for as many time steps as possible.

To solve the problem, Deep Deterministic Policy Gradient (DDPG) algorithm was applied. This is a model-free algorithm based on deterministic policy gradient that can operate over continuous action spaces Lillicrap et al. (2016) . The code was adapted from Udacity's ddpg-bipedal exercise.

Network architecture

Two main methods were trialed in this project. Method one used a single agent with batch normalisation, clipped weights and hard copying within the ddpg agent. The second method used a combination of these approaches but migrated to a twenty agent model as initial results with the single model were too noisy.

FIRST METHOD - SINGLE AGENT MODEL:

A 1D batch norm was added to the actor and critic model. Lillicrap et al. (2016) suggested to address the problem of scaling features into similar ranges by applying batch normalisation. The technique normalises each dimension across the samples in a mini batch. Which results in an improved running average mean and variance, and allow the model to work for a range of environments.

```
self.bn1 = nn.BatchNorm1d(fcs1_units)
```

Clipped weights was suggested in the udacity learnings to restrict the policy from being stuck at a non-optimal policy. This will allow it to keep improving it's policy as it lowers the error but ensure improvement in rewards.

```
torch.nn.utils.clip_grad_norm_(self.critic_local.parameters(), 1)
```

A hard copy of the weights was suggested by a udacity-tarian. This resulted in the expected results, however the score was very noisy.

```
self.hard_copy_weights(self.actor_target, self.actor_local)
self.hard_copy_weights(self.critic_target, self.critic_local)
```

The neural network architecture was very similar for the actor and critic model. Both utilised the batch normalisation method and had a three layer fully connected network. The first layer had 400 features, second layer with 300 features. It also included relu and thanh for the actor model

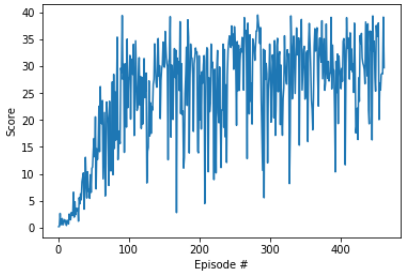
Slight changes were made to the ddpg agent hyper parameters to suit the problem.

```
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 = 2e-4         # learning rate of the actor
```

```
LR_CRITIC = 2e-4         # learning rate of the critic
WEIGHT_DECAY = 0         # L2 weight decay
```

The model achieved the required results. However the learning seemed too noisey. I suspected this may be due to the hard copying of weights and wanted to search for other methods. Such as developing a better network architecture.

```
Episode 100     Average Score: 12.22    Time Cost per epi: 8.45
Episode 200     Average Score: 26.53    Time Cost per epi: 8.55
Episode 300     Average Score: 27.83    Time Cost per epi: 8.59
Episode 400     Average Score: 29.04    Time Cost per epi: 8.57
Episode 462     Average Score: 30.03    Time Cost per epi: 8.57
Environment solved in 462 episodes!     Average Score: 30.03
```



SECOND METHOD - MULTI AGENT MODEL

This method utilised learnings from the first and made slight modifications to suit the code for a multi agent environment. It utilised

- batch normalisation in the models
- the critic had leaky relu instead of relu
- gradient clipping
- modification of hyper parameters
- reduction of units in fully connected layers from 400 → 256 and 300 → 128

The goal of migrating to a multi agent environment was to achieve a less noisy training model.

The hyper parameters used:

```
#ddpg agent lines 12-18

BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 1024       # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 5e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```

the noise for current given policy wasn't correct and it resulted in problems during further training

```
## ddpg agent 77-78

for i in range(20):
    action[i] += self.noise.sample()
```

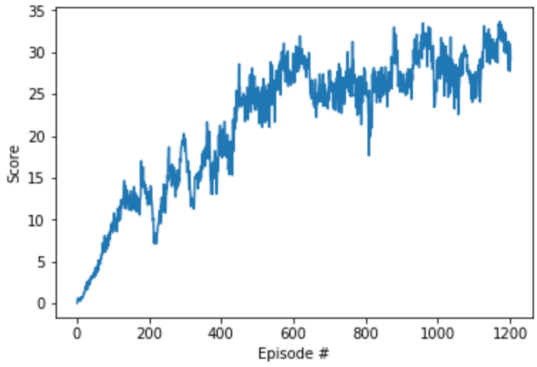
There was an attempt to add in more fully connected layers. However this resulted in poorer learning. This could be due to increasing the search space with more layers and requiring more data to learn. This could eventually get to the optimal score but with more time. Following the previous projects, less layers may be the best approach to go. Thus I reduced the units in the fully connected layers as described above.

RESULTS

The 20 agent model was able to produce stabler results than the single model agent. This is due to the increase in batch size and multiple agents acting in an episode to provide a better average result.

The agent took 1202 episodes to average a score of 30.

```
Episode 100     Average Score: 4.13     Time Cost per epi: 30.18
Episode 200     Average Score: 12.20    Time Cost per epi: 30.36
Episode 300     Average Score: 13.58    Time Cost per epi: 30.40
Episode 400     Average Score: 16.18    Time Cost per epi: 30.30
Episode 500     Average Score: 22.46    Time Cost per epi: 30.57
Episode 600     Average Score: 26.05    Time Cost per epi: 30.57
Episode 700     Average Score: 26.66    Time Cost per epi: 30.62
Episode 800     Average Score: 25.79    Time Cost per epi: 30.75
Episode 900     Average Score: 26.42    Time Cost per epi: 30.72
Episode 1000    Average Score: 28.78    Time Cost per epi: 30.89
Episode 1100    Average Score: 27.73    Time Cost per epi: 30.78
Episode 1200    Average Score: 29.93    Time Cost per epi: 30.94
Episode 1202    Average Score: 30.04    Time Cost per epi: 30.79
Environment solved in 1202 episodes!    Average Score: 30.04
```



Future work

For future improvements in this exercise, trialing algorithms such as PPO, A3c and D4PG that use distributed training methods to gain experience would be a worthwhile learning.

I did experiment with adding additional fully connected layers. This didn't result in better results unfortunately. However experimenting with hyper parameters, dropouts, weight initialisation methods to reduce the training times would be beneficial in the reinforcement environment due to its vast search space.

References

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2016). Continuous Control with Deep Reinforcement Learning. [online] Available at: https://arxiv.org/abs/1509.02971 [Accessed 9 Nov. 2018].