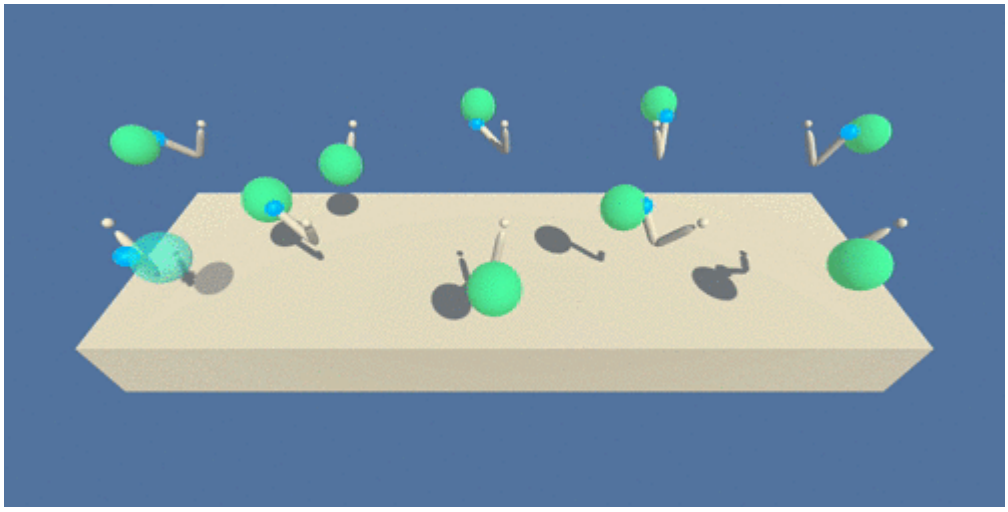
Continuous Control

Goal

The main idea of the project is to make 20 double jointed arm agents maintain their position at a target location for as many time steps as possible. In order to successfully solve the environment, an average agent score of +30 must be achieved over a period of 100 episodes.



Environment

The virtual environment is provided by Udacity and is based on Unity ML agents.

The agent gets a reward of +0.1 for each step the agent's hand is in the goal location.

The state space consists of 33 variables corresponding to position, rotation, velocity and angular velocities of the arm.

The action space is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

Learning Algorithm

In order to solve this task, a Deep Deterministic Policy Gradient (DDPG) approach has been implemented. It has been modified slightly to take into account the presence of multiple agents.

DDPG involves 2 networks. One actor and the other critic. Each of these 2 networks hold a local and a target copy of the network.

During a single step of learning,

- the local action network estimates the best action; in this continuous action case, an action vector of size 4
- the critic network uses this value to estimate the optimal action value function
- the target networks are soft updated by the local networks

Neural Network Architecture

Actor

The actor network consists of 3 fully connected layers. The initial input is passed through a batch normalization procedure, which really improved the performance of the agent.

- Layer 1: Input size = 33 (state size), Output size = 128
- Layer 2: Input size = 128, Output size = 128
- Layer 3: Input size = 128, Output size = 4 (action size)
- [State] → Layer 1 → Leaky ReLu → Layer 2 → Leaky ReLu → Layer 3 → Tanh → [Action]

```
Actor(
  (bn1): BatchNorm1d(33, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=33, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=4, bias=True)
)
```

Critic

The critic network consists of 3 fully connected layers. The initial input is batch normalized.

- Layer 1: Input size = 33 (state size), Output size = 128
- Layer 2: Input size = 132 (128 + action size), Output size = 128
- Layer 3: Input size = 128, Output size = 1 (action value)
- [State] → Layer 1 → Leaky ReLu → Layer 2 → Leaky ReLu → Layer 3 → [Action Value]

```
Critic(
  (bn1): BatchNorm1d(33, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fcs1): Linear(in_features=33, out_features=128, bias=True)
  (fc2): Linear(in_features=132, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=1, bias=True)
)
```

Note:

- There is no activation function leading to the final output layer for Critic. This is because the action values can be negative as well, but having something like a ReLu activation will only output positive values.
- The output is given by a Tanh function for the Actor network. This is because the action vector has continuous values between -1 and 1.

Hyperparameters

The following hyper-parameters has been used.

```
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-3  # learning rate of the actor
LR_CRITIC = 1e-3  # learning rate of the critic
WEIGHT_DECAY = 0  # L2 weight decay
```

In addition to these, there is a **noise decay factor** of **0.99**. A linear decay of the **noise weight** helped speed up the learning process.

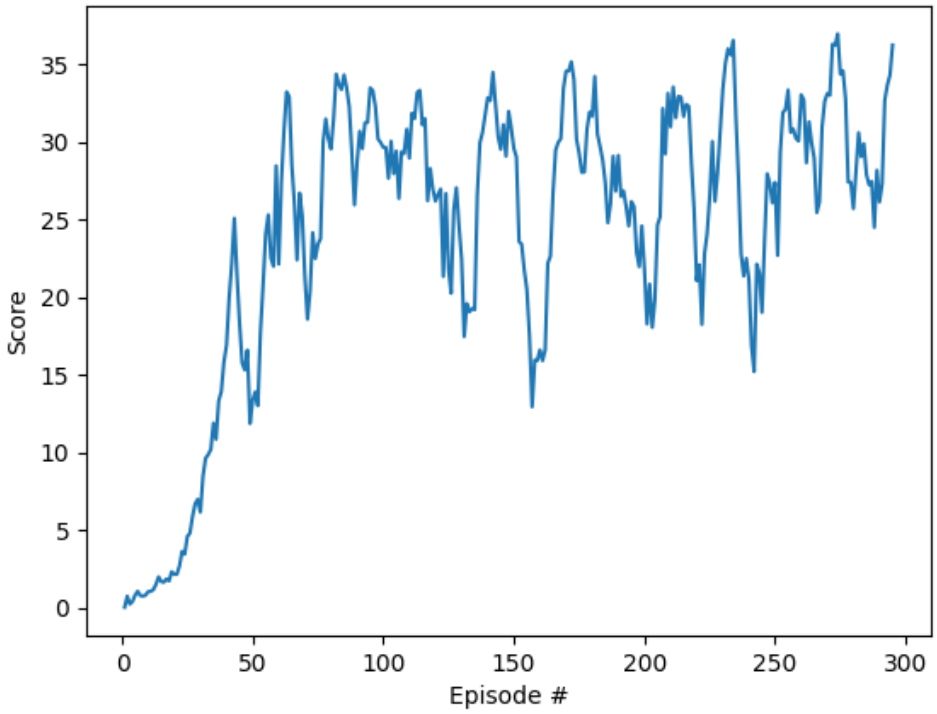
Plot of Rewards

It took 295 episodes to successfully train the agent.

```
Starting to train the agent....

Episode 8       Episode Score: 0.74     Noise Factor: 0.92      Max Score: 1.05 Average Score: 0.59
Episode 295     Episode Score: 36.24    Noise Factor: 0.10      Max Score: 36.95Average Average Score: 30.08
Environment solved in 295 Episodes      Average Score: 30.08
```

The plot is shown below.



Ideas for future work

For increased stability of the model, a few adjustments could have been done:

- Add dropout layers, which theoretically makes the network more balanced
- Reducing the frequency of the local network to target layer update could make the rewards plot smoother

Prioritized Experience Replays will definitely help improve the performance, by learning more from some experiences than others.