# DRLND – P3 Collaboration and Competition

## Learning Algorithm – Deep Deterministic Policy Gradient (DDPG)

Policy gradient methods are a type of reinforcement learning technique that rely upon optimizing parametrized policies with respect to the expected return (long-term cumulative reward) by gradient descent. They do not suffer from many of the problems that have been marring traditional reinforcement learning approaches such as the lack of guarantees of a value function, the intractability problem resulting from uncertain state information and the complexity arising from continuous states & actions.

DDPG falls into the group of policy gradient algorithms which relies on an actor-critic architecture with two elements, an actor and a critic. An actor is used to tune the parameter $\boldsymbol{\theta}$ for the policy function, i.e. decide the best action for a specific state:

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$

A critic is used for evaluating the policy function estimated by the actor according to the temporal difference (TD) error:

$$r_{t+1} + \gamma V^v(S_{t+1}) - V^v(S_t)$$

Where the lower-case *v* denotes the policy that the actor has decided. In this way the critic acts like a Q-Learning agent to learn the optimal action to take in a given state and the equation closely resembles the temporal difference (TD) learning equation from Q-Learning.

DDPG also borrows the ideas of experience replay and separate target network from Deep Q Network (DQN) learning, so there exists both 'local' and 'target' networks for both the actor and critic**.** The local networks are updated after each timestep whilst the target networks are 'soft' updated according to this equation:

$$\theta_{target} = \tau * \theta_{local} + (1 - \tau) * \theta_{target}$$

Where $\tau$ is a hyperparameter which can be set prior to training.

For this project each agent will have its own Actor and Critic networks but will be able to draw from a shared memory buffer of the experiences from both agents. This means that each agent will need to learn how to maximise their reward independently of the other agent, despite their need to collaborate (by keeping rallies as long as possible) in order to receive the maximum reward for an episode.

An issue for DDPG is that it seldom performs exploration for actions which can be mitigated by adding noise on the parameter space or the action space. One commonly used noise process is Ornstein-Uhlenbeck Random Process which is what was implemented here, along with a decaying epsilon process to reduce the amount of exploration in latter episodes.

See the next page for DDPG pseudocode and hyperparameters used during training.

## DDPG Algorithm Pseudocode:

The pseudo code and hyperparameters used for training the full DPPG algorithm are as follows:

**Algorithm 1** DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
**for** episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  **for** t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  **end for**
**end for**

## Hyperparameters

| | |
|---|---|
| BUFFER_SIZE = int(1e6) | # replay buffer size |
| BATCH_SIZE = 512 | # minibatch size |
| GAMMA = 0.99 | # discount factor |
| TAU = 0.1 | # for soft update of target parameters |
| LR_ACTOR = 1e-4 | # learning rate of the actor |
| LR_CRITIC = 3e-4 | # learning rate of the critic |
| WEIGHT_DECAY = 0.0001 | # L2 weight decay |
| UPDATE_EVERY = 1 | # how often to update the network |
| EPSILON = 1.0 | |
| epsilon_decay = 5e-3 | # decay rate for exploration |
| sigma = 0.3 | # variance for OU Noise & exploration |

## Neural Network Architectures

Each agent was initialized with the following neural networks:

The Actor neural network was comprised of two fully connected layers containing 128 hidden units each. A ReLU non-linear activation was used between each layer with batch normalization after each activation. The output layer had 4 nodes (one for each action) and a Tanh non-linearity to squash the outputs between the range [-1, 1] which was the valid range of an action for the arm across its 4 inputs.

The Critic neural network was also comprised of two fully connected layers each with 128 hidden units. The output from the first layer was passed through a ReLU non-linear activation, then through a batch normalization regularization layer and finally as input to the second layer. Here it was concatenated with the 4 actions to give 132 hidden units and passed to a single output node via a ReLU activation.

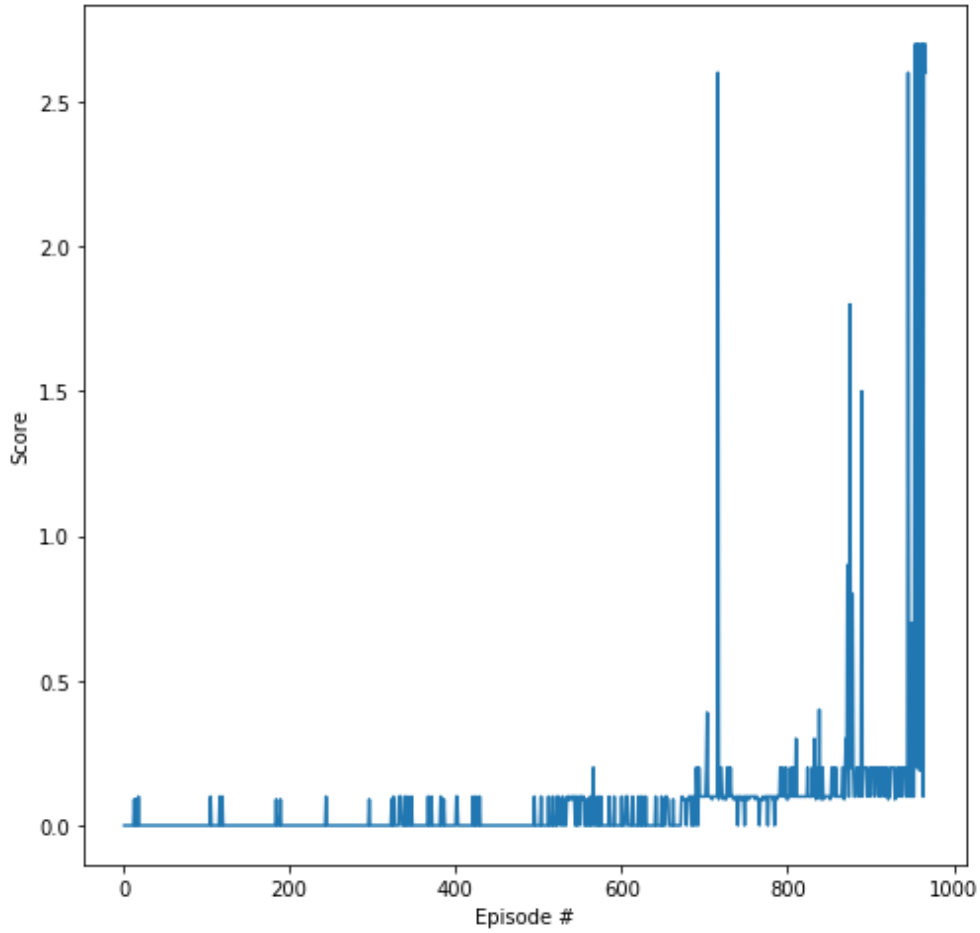
Please see appendix for a full visualisation of each network.

## Results

Using the hyperparameters mentioned above lead to the environment being solved in 866 episodes (i.e. the max discounted score across both of the 2 agents was >= 0.5 for 100 episodes).

Here is a summary of the results from training the agents:

```
Episode 100     Average Score Last 100 Episodes: 0.00280        Max Score (All Agents) Last Episode: 0.00
Episode 200     Average Score Last 100 Episodes: 0.00480        Max Score (All Agents) Last Episode: 0.00
Episode 300     Average Score Last 100 Episodes: 0.00190        Max Score (All Agents) Last Episode: 0.00
Episode 400     Average Score Last 100 Episodes: 0.01180        Max Score (All Agents) Last Episode: 0.00
Episode 500     Average Score Last 100 Episodes: 0.00600        Max Score (All Agents) Last Episode: 0.00
Episode 600     Average Score Last 100 Episodes: 0.03580        Max Score (All Agents) Last Episode: 0.10
Episode 700     Average Score Last 100 Episodes: 0.04090        Max Score (All Agents) Last Episode: 0.10
Episode 800     Average Score Last 100 Episodes: 0.13140        Max Score (All Agents) Last Episode: 0.09
Episode 900     Average Score Last 100 Episodes: 0.18340        Max Score (All Agents) Last Episode: 0.10
Episode 966     Average Score Last 100 Episodes: 0.50540        Max Score (All Agents) Last Episode: 2.60
Environment solved in 866 episodes!     Average Score: 0.50540
```




It's clear that it takes the agents a significant number of episodes to learn that collaboration is important for maximizing total reward. Is more exploration is needed in earlier episodes? Please see further discussion below.

## Future Ideas for Increasing the Agent's Performance

The limiting factor on increasing performance is the amount of time it takes to train the agent(s). If time was allowable a grid search methodology could be run allowing a range of hyperparameters (learning rates, discounting, soft update ratios etc.) to be tested so that nearer optimal parameters might be found.

In particular increasing Tau (the parameter which dictates the ratio by which the local networks are soft copied to the target networks) substantially from 1e-3 to 1e-1 reduced the number of episodes to solve the environment by approx. 50%, so it could potentially be increased further.

A clear observation is that the agents don't really make substantial increases in their score until around episode 700, which might indicate that the OU Noise process for exploration could be increased to increase the range of actions the agents take in early episodes, followed by a high epsilon to decay the noise quickly and encourage more exploitation. There is much scope to experiment with these parameters.

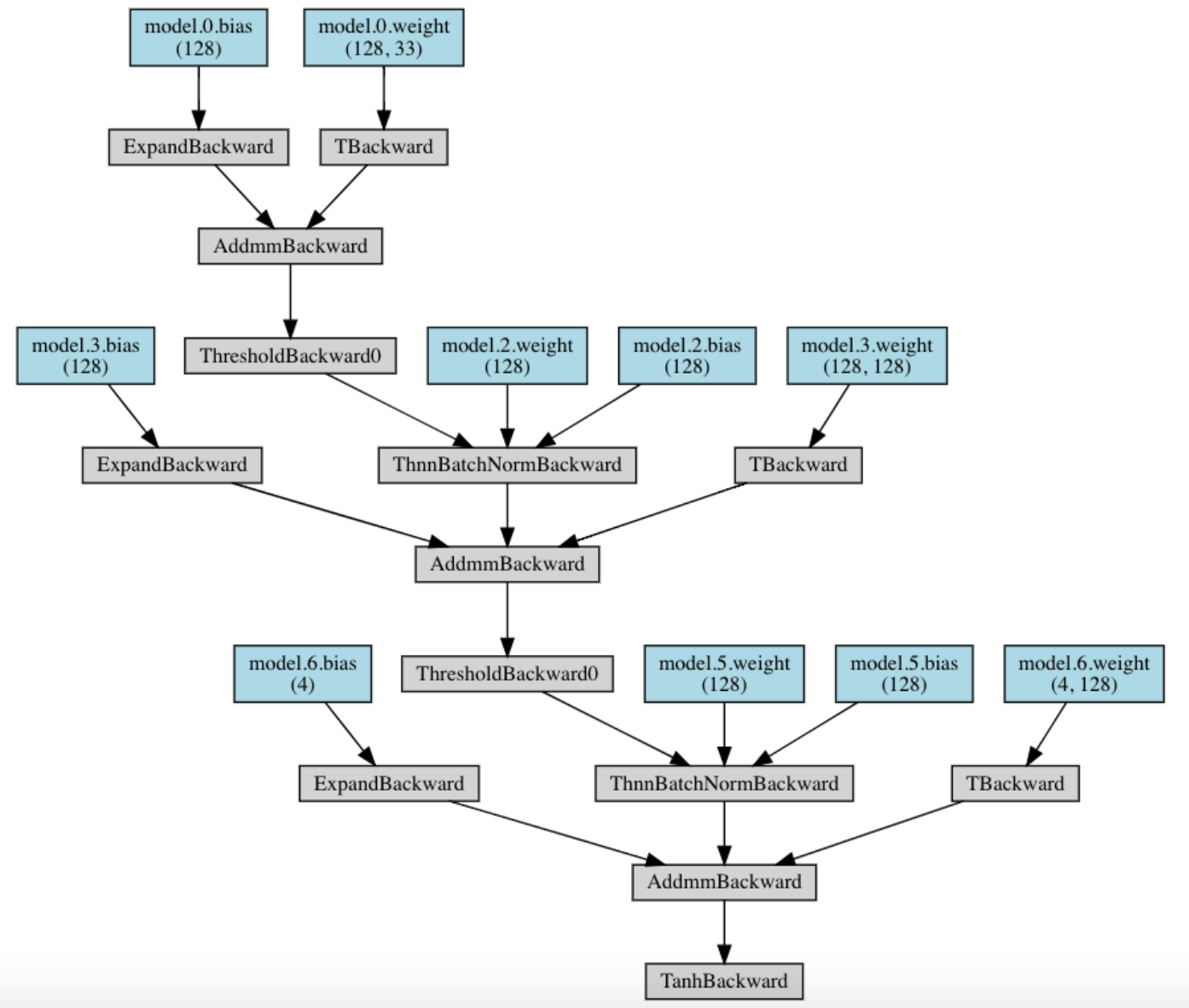
Additionally, the structures of the neural networks could be experimented with to see if adding additional layers or hidden units would allow the agent to train faster, different non-linear activations might be experimented with, or further regularizations applied.

Another recommendation might be to have a "warm up" period where the agent acts randomly to collect experiences (s, a, r, s') for a number of episodes in order to fill the experience replay memory before training begins. This would be particularly useful if prioritized experience replay were implemented also as priority would be given to sampling experiences which are likely to give the biggest improvements in the agents learning.

I also experimented with shared actor and critic networks for both of the agents, which made the project the same as the continuous control project 2. This greatly reduced the number of episodes required to solve the environment (398) as the agents both shared the best available policy, however this also means that the agents did not need to learn how to collaborate and as such was not the aim of this project.

## Appendix

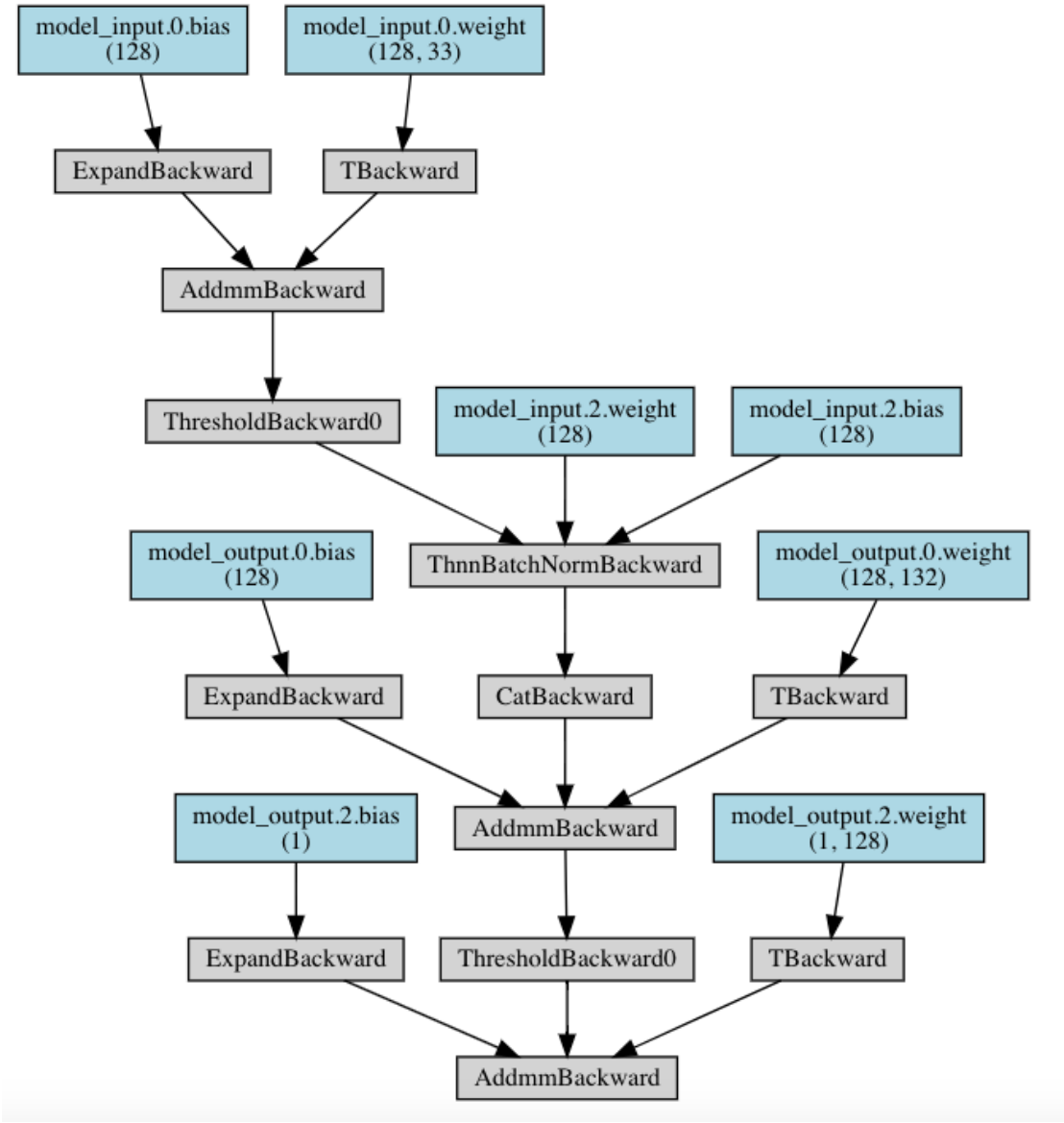
DDPG Actor Network




DDPG Critic Network

Please see the following page.

DDPG Critic Network



## References:


http://www.scholarpedia.org/article/Reinforcement_learning

http://www.scholarpedia.org/article/Policy_gradient_methods

https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

http://proceedings.mlr.press/v32/silver14.pdf

https://github.com/udacity/deep-reinforcement-learning

https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum (The basis of the DPPG algorithm used here)