Continuous Control Project Report

Implementation

The agent is trained using a modified **Deep Deterministic Policy Gradient** (**DDPG**) algorithm to solve the second version of the Reacher environment (with 20 agents).

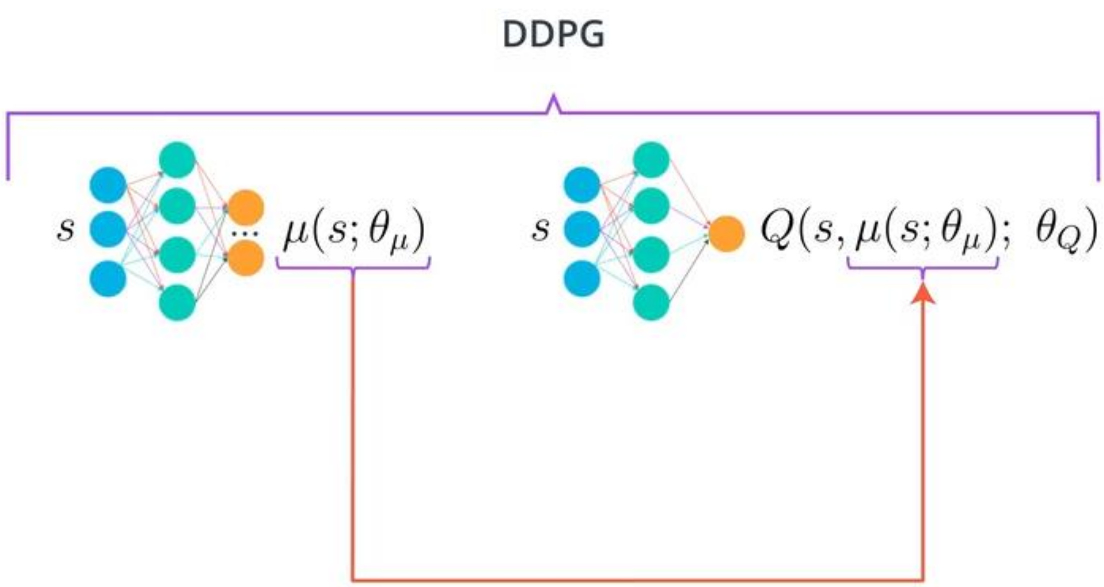
DDPG Model Architecture

The DDPG agent consists of two deep Neural Network:

1) Actor
2) Critic

The **Actor** is used to *approximate* the optimal policy deterministically, by learning $argmax_a Q(s,a)$, which is the best action.

Then the **Critic** learns to evaluate the **optimal value function** by using the Actor's best-believed action.



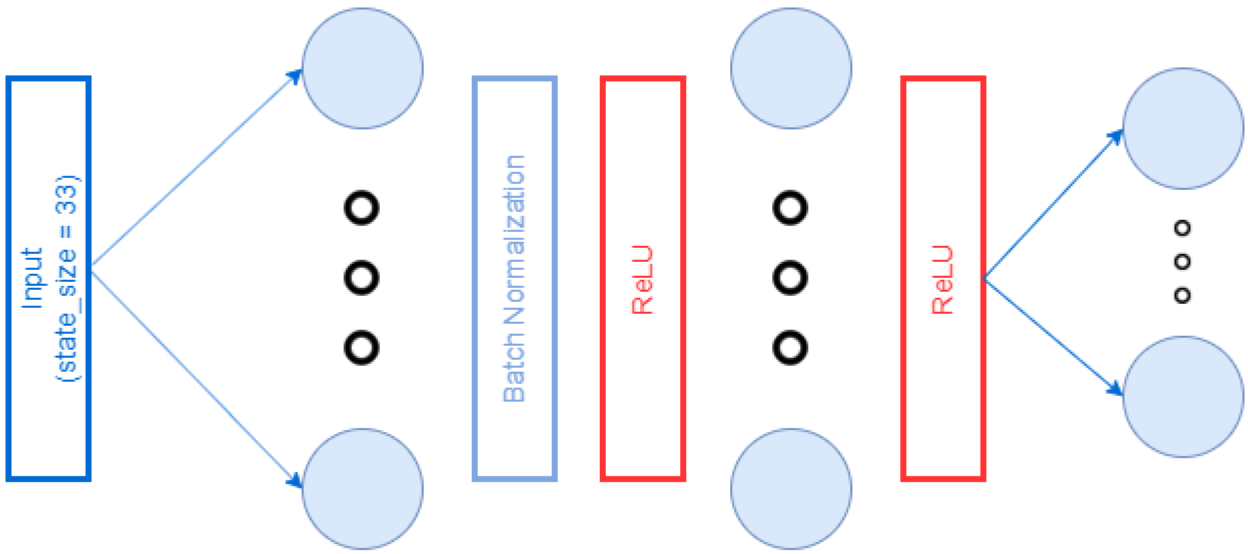
Both networks (Actor and Critic) consist of **three fully connected layers**.

The first hidden layer takes the size of the state space (*state_size = 33*) as input, and outputs *400 units*, which are passed as input to the second hidden layer.

The second hidden layer outputs *300 units*, which are passed as input to the third and final output layer.

In the **Critic**, the action size is added to the input of the second hidden layer, and the final layer outputs a single Q value.

In the **Actor**, the final output from the third output layer is the size of the action space (*action_size = 4*), corresponding to the torque applicable to the two joints.



Batch Normalization is applied to the output of the first fully connected layer to normalize each dimension across the samples in a mini-batch to have unit mean and variance and to minimize covariance shift during training.

The output of the first and second fully connected layers passes through a **ReLU activation function**.

The final output layer of the Actor is a **tanh layer**, to bind the actions.

Learning Algorithm

- Randomly initialize critic network (*critic_local*) and actor network (*actor_local*) with random weights θ^Q and θ^π.
- Initialize target networks (*actor_target*) and (*critic_target*) with weights $\theta^{Q`} \leftarrow \theta^Q$, $\theta^{\pi`} \leftarrow \theta^\pi$.
- Initialize replay memory (*ReplayBuffer*) with capacity (*BUFFER_SIZE = 10^6*).
- **for** episode (*i_episode* ← *1* to *1000*):
    - Prepare initial state: (*states* = *env_info.vector_observations*)

- Initialize *scores* ← *0*
- **for** time step *t* ← *1* to *1000*:
    - Select *actions* A from *states* S according to the current policy and exploration noise (an Ornstein-Uhlenbeck process with $\theta = 0.15$ and $\sigma = 0.2$).
    - Execute *actions* A, observe *rewards* R
    - Prepare next state: (*next_states = env_info.vector_observations*)
    - Store experience tuple (S,A,R,S`) in replay memory (*ReplayBuffer*)
    - states ← next_states
    - Add reward to score

  Every (LEARN_EVERY = *20*) timesteps:

    - Sample a random mini-batch of experience tuples (states, actions, rewards, next_states, dones) from memory (*ReplayBuffer*)
    - Compute Q targets for current states *Q_targets = rewards + (gamma * Q_targets_next * (1 - dones))*
    - Update critic by minimizing the loss and applying Gradient Clipping to avoid exploding gradient problem.
    - Update the actor policy using the sampled policy gradient.
    - Update the target networks with ($\tau = 0.001$) of the local network weights: *θ_target = τ*θ_local + (1 - τ)*θ_target*.

Hyperparameters

Variable Name	Chosen Value	Description
BUFFER_SIZE	10^6	Replay buffer size
BATCH_SIZE	128	Mini-batch size
GAMMA	0.99	Discount Factor
TAU	0.001	Soft target updates value
LR_ACTOR	0.001	Actor's Learning Rate
LR_CRITIC	0.001	Critic's Learning Rate
WEIGHT_DECAY	0	L2 weight decay
LEARN_EVERY	20	Learning timestep interval
LEARN_NUM	1	Number of learning passes

		(<u>Note:</u> using 10 learning passes as suggested in the benchmark implementation did not converge!)
GRAD_CLIPPING	1	Gradient Clipping value
OU_SIGMA	0.2	Sigma for Ornstein-Uhlenbeck noise process
OU_THETA	0.15	Theta for Ornstein-Uhlenbeck noise process

Results

```
Episode 100     Average Score: 37.64     Moving Avg.: 15.92
Episode 143     Average Score: 36.93     Time: 66s
Environment solved in 43 episodes!       Average Score: 30.18
Training complete in 144m 57s
```

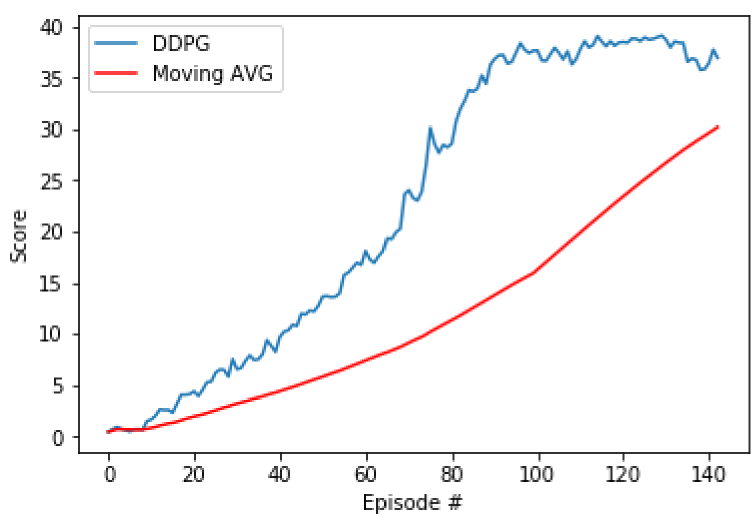


Figure 1 Plot of Rewards

The agent was able to receive an average reward (over 100 episodes, and over all 20 agents) of **30.18** in **43 episodes**.

Future Improvements

Several improvements could be implemented to enhance the agent's performance, including:

- Implementing more stable methods to achieve better performance, like: Trust Region Policy Optimization (TRPO), Truncated Natural Policy Gradient (TNPG), Proximal Policy Optimization (PPO) or the more recent Distributed Distributional Deterministic Policy Gradients (D4PG).
- Fine tuning the hyperparameters further to solve the single agent Reacher environment.

References

1) Continuous control with deep reinforcement learning

   https://arxiv.org/abs/1509.02971

2) DDPG Pendulum Exercise: Udacity's Deep Reinforcement Learning Nanodegree GitHub Repo

   https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum

3) Benchmarking Deep Reinforcement Learning for Continuous Control

   https://arxiv.org/abs/1604.06778